

---

# Development of an automatic, multidimensional, multicriterial optimization algorithm for the calibration of internal combustion engines

---

Vom Fachbereich Mathematik der Technischen Universität Darmstadt zur Erlangung des Grades eines Doktors der Naturwissenschaften (Dr. rer. nat.)  
genehmigte Dissertation von Dipl.-Phys. Timo Burggraf, geb. Fleckenstein aus Flensburg



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

Department of Applied Mathematics  
Research Group Optimization

Development of an automatic, multidimensional, multicriterial optimization algorithm  
for the calibration of internal combustion engines

Genehmigte Dissertation von Dipl.-Phys. Timo Burggraf, geb. Fleckenstein aus Flensburg

Referent: Prof. Dr. Stefan Ulbrich

Korreferent: Prof. Dr. Christian Beidl

Tag der Einreichung: 26. September 2014

Tag der Prüfung: 03. Februar 2015

Darmstadt 2015 — D 17

---

---

## Danksagung

---

Diese Dissertation wurde an der Graduiertenschule *Computational Engineering* der Technischen Hochschule Darmstadt erstellt.

Ich danke allen, die am Zustandekommen dieser Arbeit mitgewirkt haben.

Als erstes möchte ich hier meinen Betreuer Prof. Dr. Stefan Ulbrich nennen, der mit mir eine fachübergreifende Herausforderung eingegangen ist und mich mit fachlichem Rat und Expertise bei der Erstellung dieser Dissertation sehr unterstützt hat.

Mein weiterer Dank gilt meinem Korreferenten Prof. Dr. Christian Beidl, der mir half, das Problem der Optimierung aus dem Blickwinkel des Maschinenbaus zu untersuchen.

Prof. Dr. Marc Pfetsch und Prof. Dr. Michael Joswig möchte ich meinen Dank für ihre ausführlichen Ratschläge und ihre Unterstützung aussprechen.

Den Kollegen und dem Team der Graduiertenschule möchte ich danksagen, weil sie ein stets angenehmes Arbeitsumfeld für mich geschaffen haben.

Eine besondere Nennung gebührt der Firma Bertrandt GmbH, die mir die Möglichkeit gegeben hat, diese Dissertation mit einem fachübergreifenden, interessanten und zukunfts-trächtigen Thema zu bearbeiten und die mich dabei finanziell unterstütz hat. Ebenfalls danke ich der Firma Elring-Klinger Motorentechnik GmbH für die Bereitstellung der Prüfstandressourcen und den technischen Rat der Mitarbeiter.

Mein besonderer Dank gehört auch meinen Diplomanden, die zusammen mit mir an diesem Thema gearbeitet haben. Gemeinsam war es uns möglich die Eigenheiten von OpenCL zu ergründen und immer neue interessante Ansätze zu entdecken.

Abschließend möchte ich meiner Familie und besonders meiner Ehefrau und meinen Großeltern danken, die stets geduldig meinen Ausführungen zuhörten, großen Anteil an meiner Arbeit genommen und mich mit ihrem unerschütterlichen Vertrauen in mich stets bestärkt haben.





---

## Zusammenfassung

---

Die Anzahl benzin- und dieselgetriebener Fahrzeuge nimmt weltweit stetig zu, wohingegen die Verfügbarkeit fossiler Energieträger stetig abnimmt. Die verstärkte Entwicklung und Einführung von Elektrofahrzeugen wird diesen Trend in den kommenden 10 Jahren sicherlich nicht aufhalten können. Grund hierfür ist die noch stark begrenzte Reichweite reiner Elektrofahrzeuge sowie der unvollständige Ausbau von Ladestationen insbesondere in ländlichen Regionen. M. Weiss et. al. zeigt in einer Untersuchung, beauftragt durch die Europäische Union, dass obwohl die Schadstoffgrenzen für Personenkraftwagen (Pkw) kontinuierlich reduziert werden, Pkw immer noch die Hauptverursacher von Stickstoffoxiden und Kohlenmonoxid sind [JRC13, JRC13a], die sich für den Klimawandel und die Erderwärmung verantwortlich zeichnen. Im Jahre 2008 verursachten Pkw 41% der Emission von Stickstoffoxiden ( $NO_x$ ) und 24% der Emission von Kohlenmonoxid (CO) weltweit [EEA10]. Durch die Abgase von unzähligen Fahrzeugen wurden im Jahr 2009 die Grenzwerte für die Luftqualität insbesondere im städtischen Raum um 16% für Stickstoffdioxid ( $NO_2$ ) und um 26% für Ruß-Partikel ( $PM_{10}$ ) überschritten [EEA09]. Des Weiteren bewiesen P. Pelkmans und P. Debal in 2006, dass manche Modelle von Dieselfahrzeugen mit EURO 6-Zertifizierung die Emissionsgrenzen von EURO 2-4 unter realen Fahrbedingungen übersteigen [Pel06].

Um der hier geschilderten Entwicklung und den negativen Auswirkungen der Benzin- und Dieselfahrzeuge auf unsere Umwelt entgegenzuwirken, verabschiedete die Europäische Union im Jahr 2009 einen Beschluss, den etablierten Testzyklus für Pkw, den sogenannten *Neuen Europäischen Fahrzyklus* (NEFZ), durch einen realistischeren Fahrzyklus zu ersetzen, der das Fahrverhalten besser widerspiegelt, den sogenannten "Worldwide Harmonized Light Vehicles Test Procedure" (WLTP).

Die strengeren Emissionsrichtlinien, die steigenden Kosten für fossile Treibstoffe und der Kundenwunsch nach sportlichem Fahrverhalten stellen die Automobilindustrie vor große Herausforderungen. Neben der Reduktion des Verbrauches und der damit einhergehenden Reduktion des  $CO_2$ -Ausstoßes ist heute eine Vielzahl weiterer Kriterien für die Motorapplikation relevant. Hierzu zählen die Laufruhe des Motors, die thermische Belastung empfindlicher Bauteile und die Emission besonderer chemischer Verbindungen. Um den vielen Kriterien gerecht zu werden, werden moderne Verbrennungsmotoren mit einer Vielzahl von Aktuatoren und Sensoren ausgerüstet. Die Konsequenz dessen ist, dass die Applikation des Motors, also die Bestimmung der Parameter aller Aktuatoren in allen Operationspunkten, immer längere Prüfstandszeiten benötigt. Dies kostet die Automobilindustrie sowohl Geld, als auch wertvolle Zeit, bis ein neues Modell Marktreife erlangt.

In dieser Arbeit wird ein neuer, ganzheitlicher mathematischer Ansatz für die Optimierung von Motorenkennfeldern vorgestellt und diskutiert. Dieser neue Ansatz umfasst Module für die Datenerfassung, -analyse und -optimierung.

Entgegen der häufigen Verwendung der stationären Messmethodik verwendet dieser Ansatz eine quasi-stationäre Messmethodik. Im Unterschied zur stationären Messmethodik wird bei der quasi-stationären Messung auf ein Einschwingen des Systems vor der Datenaufnahme verzichtet. Bei der quasi-stationären Messmethodik wird die kontinuierliche Aufnahme von Messdaten im Motorapplikationsprozess durch eine eigenständige Verfeinerung des Datenraumes unterstützt. Das heißt, der dieser Arbeit zugrunde liegende Algorithmus entscheidet eigenständig, welche Bereiche des Datenraumes genauer untersucht werden sollen.

In dieser Arbeit werden zwei unterschiedliche Kriterien für die Datenraumverfeinerung eingeführt: die Verfeinerung aufgrund unerwarteter Abweichungen von einem lokalen Modell

---

---

und die Verbesserung der stationären Lösung. Das lokale Modell ist gegeben durch eine Polynomfunktion zweiten Grades. Die Parameter dieser Funktion werden mittels Fit an Datenpunkten der unmittelbaren Umgebung bestimmt. Die Datenpunkte in unmittelbarer Umgebung werden durch eine Voronoi-Zerlegung des Raumes ermittelt. Die Verfeinerung aufgrund des zweiten Kriteriums wird ausgelöst, wenn neue Datenpunkte die bisher bestimmte stationäre Lösung im Motorenkennfeld verbessern.

Aufgrund der größeren Anzahl der Datenpunkte innerhalb der quasi-stationären Messmethodik gegenüber der stationären Vermessung werden innerhalb dieser Arbeit zwei neuartige Filtermethoden entwickelt. Dabei handelt es sich um einen zeitlichen und einen räumlichen Filtermechanismus, welche die Daten deutlich komprimieren, jedoch die Information weitestgehend erhalten.

Als Vorbereitung auf die Kennfeldoptimierung werden in dieser Arbeit die Zielfunktion sowie alle Nebenbedingungen der ganzzahligen Optimierung formuliert. Die Zielfunktion folgt der Minimierung des spezifischen Verbrauchs. Durch die Nebenbedingungen ist es möglich, ebenfalls die Schadstoffgrenzen ausgewählter chemischer Verbindungen gemäß der Regulierung der Europäischen Union einzuhalten. Zusätzlich werden Datenpunkte, welche harte Grenzwerte überschreiten, von der ganzzahligen Optimierung ausgeschlossen. Des Weiteren werden sich stark ändernde Aktuatorstellungen innerhalb der Lösung mathematisch ausgeschlossen. Dies wird durch die Begrenzung der Steigungen innerhalb der Motorenkennfelder einzelner Aktuatoren erreicht. Dadurch kann ein Motorkennfeld erzeugt werden, das das transiente Verhalten des Motors komplett widerspiegelt.

Die aufgestellten neuen Methoden werden innerhalb dieser Arbeit vollständig validiert, sodass ihre Funktionsweise nachgewiesen wird. Hierfür werden explizite Annahmen für die Funktionsweise der Funktion aufgestellt und verifiziert.

Für einen Vergleich eigener Methoden mit etablierten Techniken wird in dieser Arbeit eine Zusammenfassung des Standes der Technik vorangestellt. Zusätzlich werden einzelne Methoden sowie der gesamte Ansatz dem heutigen Stand der Wissenschaft in Benchmark-Rechnungen gegenübergestellt. Diese Benchmarks zeigen, dass die essenzielle Bestimmung der Voronoi-Zellen durch die hier vorgestellte Methode die benötigte Qualität mit sich bringt, aber deutlich schneller als etablierte Methoden ist. Dies ist notwendig für die Sicherstellung der echtzeitfähigen Datenvalidierung.

Des Weiteren wurde das gesamte neue Konzept der Datenerfassung, also quasi-stationäre Messung, Datenvalidierung und Datenraumverfeinerung, einer typischen Rastermessung gegenübergestellt. Dieser Benchmark zeigt, dass das neue Konzept Lösungen gleicher oder höherer Qualität findet, jedoch deutlich weniger Zeit benötigt. Die Messzeit am Prüfstand gleicher Qualität konnte von 360 Stunden im Falle der Rastervermessung auf weniger als 40 Stunden reduziert werden.

Die Wirkweise und die Qualität der gesamten Motorenvermessung und Optimierung werden in dieser Arbeit mit Hilfe zweier Motorenmodelle getestet. Das erste Motormodell basiert auf einer Simulation eines Benzinmotors mit variablen Ventilsteuerzeiten und Saugrohreinspritzung. Dieses empirische Modell wird ebenfalls im Zuge dieser Arbeit entwickelt - primär als Datenquelle für die Entwicklung des Hauptprogramms. Das zweite Modell wird durch eine Kooperation mit der Firma AVL GmbH in das Gesamtsystem integriert. Hierbei handelt es sich um ein validiertes Motorenmodell eines Dieselmotors mit Turbolader, einer Voreinspritzung und variabler Ladergeometrie. Diese Motorenmodelle werden in gleicher Weise wie ein Motorenprüfstand verwendet. Das setzt voraus, dass das synthetische Signal um einen statistischen und systematischen Fehler erweitert wird.

---

Anhand dieser Motorenmodelle wird eine komplette Optimierung beider Systeme durchgeführt und diskutiert. Diese Arbeit zeigt, dass der neue Ansatz die gewünschten Ergebnisse erzielt. Beide Modelle können bezüglich spezifischen Verbrauchs optimiert werden. Die Nebenbedingungen des Optimierproblems werden dabei erfüllt: also die Einhaltung aller harten Grenzen sowie der maximalen Emissionswerte der EU und die Wahrung aller maximalen Gradienten im Lösungsfeld. Diese Optimierungen werden auf Basis zweier Testzyklen durchgeführt: dem NEFZ und dem neu eingeführten WLTP.

Diese Arbeit zeigt, dass die Testzyklen einen sehr großen Einfluss auf die Lösung der Kennfeldoptimierung haben. In Folge dessen empfiehlt es sich, die Testzyklen stärker realistischen Fahrsituationen anzupassen. Hauptaussage dieser Arbeit ist, dass das Potenzial der Optimierung moderner Verbrennungsmotoren noch lange nicht ausgeschöpft ist. Insbesondere die Behandlung mehrerer Kriterien für die Optimierung zeigt auf, dass niedrigere Emissions- und geringere Verbrauchswerte möglich sind. Im Besonderen die Bestimmung der transienten, also fahrbaren Kennfelder, anstatt stationärer reduziert den Applikationsbedarf im gesamten Fahrzeugentwicklungsprozess und liefert verlässlichere Kennfelddaten.

---

## Contents

---

<b>1</b>	<b>Introduction</b>	<b>8</b>
1.1	Description of the data acquisition and optimization problem . . . . .	9
1.2	Outline of this work . . . . .	12
<b>2</b>	<b>Theory</b>	<b>15</b>
2.1	Mathematics . . . . .	15
2.1.1	Random Walk . . . . .	15
2.1.2	Support Vector Machine . . . . .	16
2.1.3	Evolutionary Algorithms . . . . .	19
2.1.4	Linear Integer Programming . . . . .	21
2.1.5	Voronoi Diagrams . . . . .	27
2.1.6	Linear Regression . . . . .	29
2.1.7	Shortest Path Problem . . . . .	30
2.1.8	Dijkstra Algorithm . . . . .	30
2.2	Engineering . . . . .	32
2.2.1	Actuators . . . . .	32
2.2.2	Sensors . . . . .	34
2.2.3	Emission Regulation and Quantification Cycles . . . . .	35
2.2.4	Stationary, Transient and Quasi Stationary Measurement . . . . .	39
2.3	Applied informatics . . . . .	41
2.3.1	Binary Heaps . . . . .	41
2.3.2	Beneath-and-Beyond . . . . .	42
<b>3</b>	<b>Calibration Methods - State of the Art</b>	<b>44</b>
3.1	Commercial Solutions . . . . .	44
3.1.1	Determination of data boundaries . . . . .	44
3.1.2	Test run planning . . . . .	44
3.1.3	Measurement method . . . . .	45
3.1.4	Models . . . . .	46
3.2	Preceding Stages of this project . . . . .	47
3.2.1	Random Walker . . . . .	47
3.2.2	Deterministic Walker . . . . .	49
3.2.3	Greedy Ants . . . . .	51
<b>4</b>	<b>Framework for the quasi stationary data acquisition and integer optimization</b>	<b>57</b>
4.1	General . . . . .	57
4.1.1	TCP/IP Server . . . . .	57
4.2	Data Acquisition and Cleaning . . . . .	58
4.2.1	Data Space and Solution Field Organization . . . . .	59
4.2.2	Dynamic Test Drive . . . . .	60
4.2.3	Measurement routing . . . . .	62
4.2.4	Data Cleaning . . . . .	64
4.2.5	Measurement Specialization . . . . .	66
4.2.6	Virtual Test Bench . . . . .	69

4.3	Data Preparation and Compression	75
4.3.1	Time Series Compressor	75
4.3.2	Adaptive Space Compressor	75
4.4	Data Analysis and Discretization	79
4.4.1	Determination of nearest Neighbors	79
4.4.2	Determination of Tolerance Table	90
4.4.3	Definition of Linear Integer Optimization Problem	91
<b>5</b>	<b>Results</b>	<b>94</b>
5.1	Validation of Methods	94
5.1.1	Dynamic Test Drive	94
5.1.2	Dijkstra Algorithm	97
5.1.3	Data Cleaning	100
5.1.4	Controller Oscillation Filter	102
5.1.5	Measurement Specialization	103
5.1.6	Adaptive Space Compressor	108
5.1.7	Determination of nearest Neighbors	110
5.1.8	Determination of Tolerance Table	114
5.2	Benchmarks	116
5.2.1	Voronoi Tessellation	116
5.2.2	Stationary Solution	117
5.3	Evaluation of preceding stages of AmmOC	122
5.3.1	Random Walker	122
5.3.2	Deterministic Walker	127
5.3.3	Greedy Ants	130
5.4	Optimization Results	135
5.4.1	Transient Optimization of AVL Engine Model	135
5.4.2	Transient Optimization of VTB	149
<b>6</b>	<b>Summary and Outlook</b>	<b>158</b>
<b>7</b>	<b>Appendix</b>	<b>161</b>
7.1	Quick Start Manual	161
7.2	Configuration of the reference problems	168
7.2.1	Simulation Model VTB	168
7.2.2	DoE Model AVL	169

---

## 1 Introduction

---

On June the 10th in 2013, 1,102,815,060 cars existed at 11:23 AM. One minute later this number increased to 1,102,815,213 [Wor13]. These cars are not distributed equally on this planet. Most cars are registered in Europe; the highest number of cars per head are in the USA [Die07]. 775 cars are divided between 1000 inhabitants in the US; in Germany the same amount of people share 573 cars.

It can be assumed that in 2050, the total number of registered cars will have increased to 2.7 billion [Wor13]. Even with respect to the extrapolation error, the systematical trend is obvious: the number of fossil fuel consumers and the number of exhaust emitters will increase noticeably. The Chinese market will be of special interest. In 2007, 22 cars were shared between 1000 Chinese inhabitants. The prognosis of this market is above average. Nearly all car manufacturers expect a great demand of cars in this market, so they invest in additional car production lines in this country.

The principle of the modern car engine was invented at the end of the 19th century. The development of internal combustion engines was carried out by different inventors, Joseph Étienne Lenoir, Nikolaus August Otto and Gottlieb Daimler<sup>1</sup>. But all share the same principle: the consumption of fuel in order to convert chemical energy into rotational kinetic energy.

This principle of the four cycle engine is still applied in modern car engines, but due to the increasing costs of fossil fuels, more stringent maximal emission limits and even more challenging customer preferences the internal combustion engine has become more and more complex. In order to combine customer requests relating to 'driving pleasure'<sup>2</sup> and the regulations of the European Union, the state-of-the-art engines are equipped with various new actuators and sensors. New techniques, e.g. direct fuel injection, turbochargers, compressors and camshaft phase shifts, cause new problems: direct fuel injectors promote the formation of soot, turbochargers and increasing compression ratios cause the emission of nitrogen oxides and high temperatures on devices.

In general, the successful calibration of an engine has to meet a large number of objectives and constraints. Besides an ideal specific fuel consumption, the creation of exhaust gases, e.g.  $\text{NO}_x$ , CO, HC,  $\text{PM}_{10}$ , the engine smoothness, the noise of combustion and the structural component stress have to be minimal or at least as small as necessary. It is important to consider that the identified optimal solution has to represent a physically meaningful setting of the actuators, i.e. physical constraints, e.g. physical reaction times, have to be fulfilled. Due to this list of requirements, the measurement process and the optimization process become challenging.

This work focuses on advanced calibration methods for internal combustion engines. Since the number of registered cars cannot be influenced, we focus on the consuming and emitting multipliers, the total fuel consumption and the total exhaust emissions. In this work, we have developed a new holistic approach for the measurement, analysis and optimization process for the application of an internal combustion engine. Moreover, this work discusses the introduction of new emission regulations and its impact on the engine application process.

M. Weiss et al. [JRC13, JRC13a] analyzed on-road emissions of light-duty vehicles and the introduction of complementary emissions tests for light-duty vehicles. These studies are mandated by the European Commission. Even if the emission limits were reduced in

---

<sup>1</sup> <http://www.kfz-tech.de/Ottomotor.htm>

<sup>2</sup> BMW commercial

the past decade, road transport is still the largest source of nitrogen oxides and carbon monoxides. The fractions of the total emission within the European Union in 2008 were 41% and 34%, respectively [EEA10]. The applicable air quality standards [EEA09] were exceeded in 2009 by  $NO_2$ , 16% and  $PM_{10}$ , 26%. New emission regulations by the European Union led to more stringent emission limits for light-duty vehicles with Euro 5b in 2011 and Euro 6 in 2014 [EC07, EC08]. Furthermore, the European Union announced a new test procedure that replaces the 'New European Driving Cycle' NEDC [EC09]. A large discrepancy has been determined between NEDC in laboratory conditions and actual on-road emissions of light-duty vehicles. That is why the European Union studies alternative test procedures that replace the NEDC. These new test procedures have to reflect real-world driving profiles in order to increase the prognosis accuracy [Art04, JRC13]. L. Pelkmans and P. Debal proved that the on-road  $NO_x$  emissions of several Euro 6 light-duty diesel vehicles exceed Euro 2 - 4 emission limits [Pel06] in the test procedure.

This work discusses the impact of the test procedure on the generated engine maps. Therefore, we compare the application results of both test cycles. So we quantify the influence of the test cycles on the application results. Finally, this work studies the compatibility of calibrations for different driving patterns.

---

## 1.1 Description of the data acquisition and optimization problem

---

An internal combustion engine varies its actuators dynamically, in order to generate the requested engine torque and revolution frequency. Both these parameters are requested by the car driver. A detailed description of the most prominent actuators is given in Subsection 2.2.1. Generally, these actuators influence the initial conditions of the combustion of the fuel-air mixture in the combustion chamber.

Besides the generation of the requested engine torque and revolution frequency, the engine has to meet several objectives and side conditions. The best known objective is a low specific fuel consumption. An ideal specific fuel consumption is given if the engine consumes the smallest amount of fuel possible while generating the requested engine power. Typical side conditions are given by a maximal emission of chemical pollutants and hard thermal bounds for devices.

Principally, we want to optimize the characteristic engine maps of internal combustion engines. A characteristic engine map, given as set  $M$  of matrices  $m_{act} \in \mathbb{R}^{n \times m}$ .  $n$  rows are given by a series of revolution frequencies  $\nu$ ,  $m$  columns are given by the series of engine torques  $M$ . The actuator settings  $m_{act}$  depend on the operation point

$$O_\nu^M \in \mathbb{R}^2, O_\nu^M = (\text{Revolution frequency, Engine torque}), \quad (1)$$

which is given by a revolution frequency and engine torque combination.

Table 1 is a typical example of one of the engine maps, which is flashed to the engine control unit ECU.

In this example, the parameter set-value for the injected fuel amount is given in accordance with the current revolution frequency and engine torque. These kind of maps have to be stored to the ECU for all dynamic actuators of the internal combustion engine, e.g. air valve angle, spark angle, CAM shifts. Furthermore, we store measured sensor values of parameters of interest in the same structure. These sensor values are used in order to predict fuel consumption and exhaust emissions of certain driving profiles.



**Table 1:** Symbolic example of an engine map

Injected Fuel [mg/cycle]	Torque [Nm]			
	4	12	...	125
Revolution frequency [1/min]				
650	4.5	8.7	...	9.2
1500	4.8	9.0	...	8.7
...	...	...	...	6.2
6250	7.8	10.2	...	12.1

In order to derive the optimal engine settings, we have to measure the engine behavior at engine test benches. These engine test benches are equipped with additional sensors. A detailed overview on the most important engine sensors is given in Subsection 2.2.2.

During the data acquisition we gather data points  $d_i \in D, i \in [0, \dots, n]$ , where  $n$  is the number of measured data points. These data points are given as vectors in  $\mathbb{R}^n$ . Each vector  $d_i$  contains *dynamic* actuator settings  $A_d$ , *static* actuator settings  $A_s$  and measured *sensor* values  $M$ .

$$d_i = (A_d^0, A_d^1, \dots, A_d^{j-1}, A_s^0, A_s^1, \dots, A_s^{k-1}, M^0, M^1, \dots, M^{l-1}) \in \mathbb{R}^{j+k+l} \quad (2)$$

An exemplary data entry is:

$$d_i = \left[ 1250 \frac{1}{s}, 12^\circ CA, 0^\circ CA, 299 \frac{g}{kWh}, 765K \right] \quad (3)$$

This simple example contains the dynamic actuators revolution frequency  $\nu_{CA}[\frac{1}{s}]$  and air valve angle  $\alpha_{air}[^\circ CA]$ , the static actuator inlet cam phase angle  $\phi_{in}[^\circ CA]$  and the sensor values specific fuel consumption  $b_e[\frac{g}{kWh}]$  and turbocharger temperature  $T_{exh}[K]$ . The whole data set  $D$  is defined as the set of all vectors  $d_i$ . In the following paragraph, we use the term *data point* for measured data points, but also for artificially calculated data points which have not been measured. Artificially calculated data points will be used in the context of *Support Vector Machines*.

The optimization of the engine maps of an internal combustion engine has to satisfy several side conditions. In this work, we focus on the most important constraints for base emission calibrations. In the base emission calibration, one determines the basic settings of the engine for all operation points. The exhaust emissions are measured behind the turbine of the turbocharger, but before the exhaust gas passes the exhaust gas after treatment, e.g. oxidation catalyst, SCR catalyst, etc. So, one tries to minimize the production of exhaust gas, the specific fuel consumption, the engine noise level and other criteria. Typically, the final application of engines includes the definition of operation strategies, e.g. exhaust after treatments, re-generation cycles of catalyst and special operation modes. These special functions for operation strategies are not part of this work.

All data points  $d_i \in D$  are grouped with respect to their location in the solution. A stack or operational point  $s_{kl} \in S$  defines a rectangle region in the solution map.  $k$  defines a certain interval of revolution frequencies  $\nu := [b_k^\nu, b_{k+1}^\nu]$ ,  $l$  defines an interval of engine torque  $M := [b_l^M, b_{l+1}^M]$ . So a stack  $s_{kl}$  can be defined as:

$$s_{kl} := \{x \in \mathbb{R}^n | x \in [b_k^\nu, b_{k+1}^\nu] \text{ and } x \in [b_l^M, b_{l+1}^M]\} \quad (4)$$



Data points  $d_i$  are grouped in a stack  $s_{kl}$  if its revolution frequency and its engine torque are within the given intervals of the stack  $s_{kl}$ .

Furthermore, the optimization has to satisfy several constraints which are introduced in the following. The solution, i.e. the optimal engine map, must not include data points that violate lower or upper bounds in order not to damage the engine or the equipment. Let  $d_i^j$  be component  $j$  of data point  $d_i$ . Furthermore,  $\overline{p_j}$  is the upper limit of component  $j$ ,  $\underline{p_j}$  is the lower limit of component  $j$ . So,  $d_i^j$  has to be within the lower and upper bound for all  $j$  and for all data points, which are part of the solution. Formally written for all  $i$ , which contribute to the resulting engine map:

$$d_i^j \in [\underline{p_j}, \overline{p_j}] \forall j. \quad (5)$$

The next constraint, the maximal integral emission constraint, defines an upper limit for the production of exhaust emissions in a test cycle. The detailed definition of test cycles is given in Subsection 2.2.3. In principle, each data point  $d_i \in D$  in the solution is related to a mean resistance time in the test cycle  $\delta t_i$ . This mean resistance time is correlated to a weighting factor  $u_i$ , which shows the partial contribution of data point  $d_i$  to the exhaust emission value. The set  $E$  is given by indices that point to the components in  $d_i$ , which contain emission sensor values. The limit value  $\overline{d_j}$  of exhaust species  $j$  has to be compared to the time integral value

$$\int_{t'=0}^{t'_1} u_i(t) \cdot d_i^j dt \leq \overline{d_j} \forall j \in E \quad (6)$$

where  $t'_1$  is the duration of the test cycle for all  $j \in E$  and for all  $i$  that are part of the solution.

The following constraint has to handle the driveability of the solution. A solution is driveable, if the resulting engine map does not contain bigger slopes than a defined upper limit  $\nabla_{max}^m$ . The solution surface or manifold  $S$  is given by an user selection of *dynamic* and *static* actuator variables and *sensor* variables  $S \subseteq \mathbb{R}^{j+k+l}$ . The slope of *dynamic* actuator  $m$  in the solution surface  $\nabla_{ij}^m$  is given by the euclidean distance function

$$\text{dist}_S(d_i, d_j) = \sqrt{\sum_{k \in S} (d_i^k - d_j^k)^2} \quad (7)$$

and the difference of the *dynamic* actuator value of point  $i$  and  $j$ :

$$\nabla_{ij}^m = \frac{|d_i^m - d_j^m|}{\text{dist}_S(d_i, d_j)} \leq \nabla_{max}^m \quad (8)$$

The final constraint has to be defined in order to acquire complete engine maps. Therefore, we define a penalty value, in the case that a stack  $s_{kl}$  cannot contribute to the solution. A stack can be empty, due to a lack of measurement points in this region, or all remaining data points in  $s_{kl}$  may violate at least one defined constraint. Furthermore, we define that exactly one data point can be chosen to be part of the solution. If zero points have been chosen, one has to add the upper limit value of each exhaust emission value to the integral exhaust emission constraint (see Equation (6)).

---

The objective function describes which aspect has to be maximized or minimized. In our specific case, we want to minimize the specific fuel consumption in each sub region of the solution. In order to compare data points with each other, one defines a *prey* value  $p_i$ . Therefore, we compare the smallest measured value of the specific fuel consumption  $b_e$  of all data points that are members of the same stack to data point  $d_i$ . Let  $\min(s_i)^{b_e}$  be the smallest value of  $b_e$  of the stack that contains data point  $d_i$ ; furthermore  $d_i^{b_e}$  is the measured value of  $b_e$  of data point  $d_i$ . So the *prey* value of  $d_i$  is given by

$$p_i = \frac{\min(s_i)^{b_e}}{d_i^{b_e}}. \quad (9)$$

---

## 1.2 Outline of this work

---

In this work, we develop methods to acquire, clean, analyze and optimize data taken from an internal combustion engine. In order to realize these functions, the algorithm has to be subdivided into several modules, *measurement plan creation*, *data acquisition*, *data cleaning*, *data space description*, *data analysis*, *data quantification* and finally *optimization*.

The creation of measurement plans before a measurement is necessary to obtain reasonable data. These plans contain information, which sensor has to be logged and which actuator- and control parameter have to be varied. Furthermore, measurement duration is defined for each signal and the measurement structure is defined, in order to obtain a good coverage of the data space. A direct modification of the measurement plan is needed (Online- capability). Thus measurement plans can be refined or coarsened, in order to save measurement time.

During the measurement run, the measurement plan is executed. The variation of parameters is done by actuators, which are directly mounted on the test engine. Actuators are typically, servomotors, valves, breaks, pumps, relays and much more. These devices can be actuated or controlled. The detection of observables is realized by sensors, i.e. temperature, pressure, revolutions and chemistry. All actuators and sensors are connected to the measurement computer, typically controlled by a specific measurement operation system, e.g. INCA, Puma. The measurement time per data point indicates the measurement method. One separates between stationary and transient measurement. Stationary measurements hold parameter combination for several minutes, before the observables are stored. Transient measurements often do not have a defined holding time. During automatic measurements, it is very important to preserve data room boundaries, in order not to damage the test engine. This is done by, keeping indicating observables in well-known limits.

Critical for the creation of an optimal operation map of an ICE is a reasonable quantification of the measured data. Therefore, unreasonable data and measurement errors, due to arbitrary controller behavior or user input errors, have to be detected and erased. Additionally, measured data has to be weighted locally and globally, with respect to several objectives, thus good and bad parameter settings can be separated. To be desirous to save measurement time, we use mathematical or physical models to predict observables. Furthermore, data analysis and models are used to define the description quality.

Time series of data points are combined to optimal solutions after the quantification. The optimization algorithms separate between boundary condition-free and constrained operation maps. Boundary condition-free maps contain parameter settings with respect to operational points, this means the setting of actuators, e.g. CAM shift settings depending on

---

several combinations of revolutions and torque (operational points), in order to achieve optimal fuel consumption and other objectives. However, these operational maps cannot be applied in real ICE, since necessary boundary conditions are not conserved. The constrained solution contain additional information about maximal gradients of several parameters and additional conditions, which have to be conserved, and therefore can be applied on real engines. These boundary conditions are given by the physical system, which does not react instantaneously. For example, the Start of Injection (SOI) parameter, defines the start of fuel injection into the burning chamber. The variation of this timing can be done instantaneously, but the reaction of the engine takes some time, so that the effect is visible in the observables.

The form of the solution field is one more important fact. Classically, the operational map is spread between engine revolutions and engine torque. More complex engines does not only contain more actuators, but need more complex operational maps. One separates between highly dimensional and dynamic operational maps. Highly dimensional maps contain more than just revolutions and torque, i.e. temperature of the catalyzer. Dynamic operation maps are a specific case of highly dimensional maps, which contain the time after setting one operational point, too.

This work discusses new holistic approaches, that solve the optimization problems of modern internal combustion engines. So, this work includes former approaches, which have been processed by three diploma theses. In order to improve comprehensibility this work introduces theoretical basic knowledge in Section 2. Since this section treats former approaches, too, it introduces the mathematical definitions of former solution approaches. The first method is based on *Random Walks*, which are discussed in the theory Subsection 2.1.1. The derived heuristic method is explained shortly in Subsection 3.2.1 and a comparison of the optimization results to this work is given in the result Section 5.3.1. The detailed discussion of this approach may be found in the diploma thesis of M. Mertz [Mer10].

The second method is based on support vector models. The mathematical basics are introduced in Subsection 2.1.2. Based on these mathematical principles M. Gebhard developed in his thesis the *Deterministic Walker* algorithm. This model-based approach is introduced in Subsection 3.2.2. A detailed description is given in the thesis of M. Gebhard [Geb11]. The comparative studies to this work are given in the Subsection 5.3.2.

The third method is the first approach, which is able to solve the whole mathematical optimization problem as defined in Subsection 4.4.3. This approach is based on the principles of evolutionary algorithms. The class of evolutionary algorithms is discussed in the theory Subsection 2.1.3. The developed algorithm *Greedy Ants* by P. Lind is explained shortly in Subsection 3.2.3. This optimization approach is model-free. A detailed discussion of this method and optimization results are discussed in [Lin12]. Excerpts of the results of the evolutionary approach are compared to results of this work in Subsection 5.3.3.

The work contains the introduction of theoretical basic knowledge (see Section 2) in mathematics (see Subsection 2.1), engineering (see Subsection 2.2) and informatics (see Subsection 2.3). These subsections contain the theoretical basics, which have been applied in this and in former approaches. The Subsection 2.2 introduces both optimization scenarios. This involves the definition and explanation of all actuators and measurands of the data models. Furthermore, we introduce emission regulations given by the European Union. The next section is a summary of common and state of the art methods to solve this class of problems (see Section 3). We introduce commercial software products, which are applied for engine application and we present preceding stages of this work. In this brief description, we point out the differences of commercial methods and our approach. In Section 4 we

---

present and discuss the new methods and new functions, which have been developed in this work. Furthermore, we provide theoretical background information, which are validated in the the result section. The final section contains the results of the applied methods. Section 5 presents the validation (see Subsection 5.1) of the methods of Section 4. Furthermore, this section contains benchmark calculations, in order to compare new methods to state of the art methods (see Subsection 5.2). Finally, we present and discuss the optimization results of two complete engine map optimization runs of a modern diesel and a gasoline car engine.

---

## 2 Theory

---

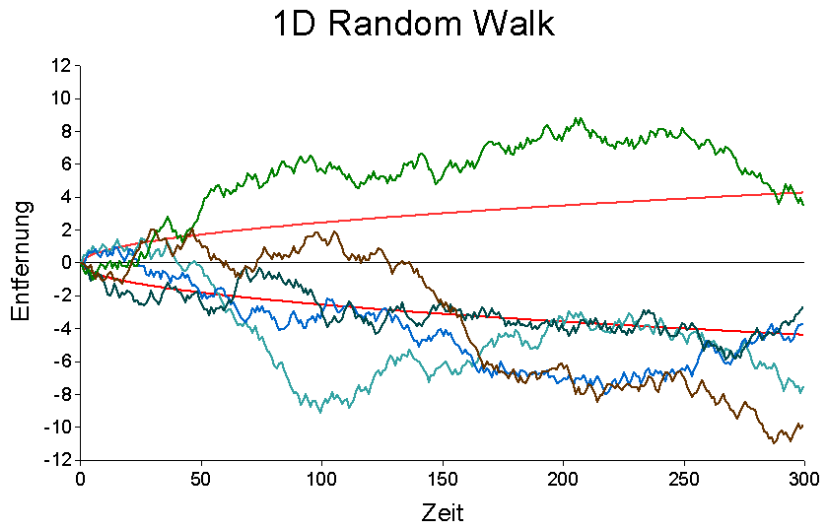
### 2.1 Mathematics

---

#### 2.1.1 Random Walk

---

A *Random Walk* is a mathematical approach to model stochastic processes [Mer10]. The term *Random Walk* has been mentioned first by Karl Pearson in 1905 [Pea05]. The very basic theorems have been contributed by George Pólya in 1921 [Pól21]. In general, a one dimensional *Random Walk* can be understood as discrete limited Markow-chains [Kon09].



**Figure 1:** one-dimensional *Random Walk*, Markow chain<sup>3</sup>

*Random Walk* are used to model financial stochastic processes, for example fluctuating stock (Random Walk Theory) or in order to study probability distributions of physical observables [Hee05].

The definition of random walks are taken from [Spi01] and [Hee05].

Define lattice points  $X_i \in \mathbb{R}^d$  of a given graph as d-dimensional integers. Formally, random walks can be defined as stochastic process  $(X_n)_{n \in \mathbb{N}}$  in  $\mathbb{R}^d$ . Assume a sequence of independent random variables  $(R_1, R_2, \dots, R_n) \in \mathbb{R}^d$ .

$$X_n = X_0 + \sum_{j=1}^n R_j, n \in \mathbb{N}_0 \quad (10)$$

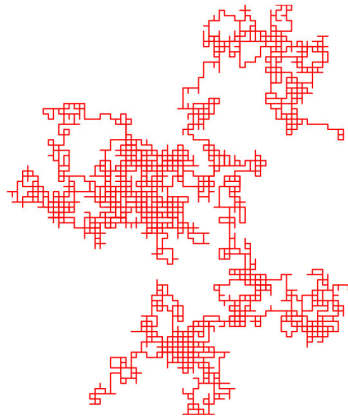
For a pair  $X_i$  and  $X_j$  in  $\mathbb{R}^d$  one defines a function of real numbers  $P(X_i, X_j)$  as *transition function* of the random walk.

The main characteristic of a *Random Walk* is it's independence of every single step of previously taken steps. So, the stochastic processes depend only on the current position. Therefore, it is possible that a *Random Walk* reaches points, which have already been crossed (see

---

<sup>3</sup> Source: <http://upload.wikimedia.org/wikipedia/de/7/71/Randomwalk1rp.png>

Figure 1). Additionally, *Random Walk* paths can be crossed by themselves. *Random Walks* can be separated by two criteria, symmetry and step width/ path structure. A symmetric *Random Walk* means, that the probability of choosing a random variable  $R_j$  is always equal. Therefore, a symmetric one-dimensional *Random Walk* has it's expectation value / final position on it's start position [Woe00]. That is why, these kind of *Random Walks* are called *recurrent*. An asymmetric *Random Walk* do not have equal probabilities, furthermore, the probabilities of choosing  $R_j$  does not have to be constant in time. The second criterion step width / path structure, divides *Random Walks* into three groups, constant step width, non-constant step width and a special case of constant step width so called lattice *Random Walks* (see Figure 2).



**Figure 2:** *Random Walk* in two dimensions<sup>4</sup>

M.Mertz [Mer10] applied the principle of random walks in the first heuristic algorithm. A introduction of this algorithm is given in Subsection 3.2.1.

### 2.1.2 Support Vector Machine

Support Vector Machines SVM are supervised learning models that analyze and classify data [Geb11, Bas07]. The basic concepts of SVM have been developed by V. Vapnik [Vap74] while working on a separable bipartition problem at the AT & T Bell Laboratories. The general idea of this algorithm is to perform a discriminative classification of test data, afterwards predict classification of unseen new data points. The initial work has been focused on OCR (optical character recognition) [Bas07].

Assume training data defined by:

$$(x_1, y_1), \dots, (x_m, y_m) | x_i \in \mathbb{R}^p, y_i \in \{-1, 1\} \quad (11)$$

$y_i$  is the corresponding classification value of test data point  $x_i$ . Classification has to be  $-1$  or  $1$ .

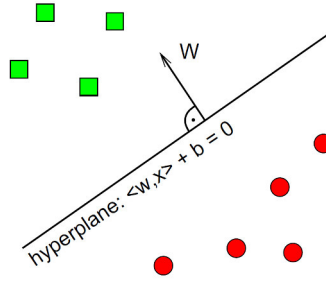
The SV algorithm has to determine a hyperplane  $h$ , so that all test data points  $x_i$  are separated by  $h$  to their classification  $y_i$ . The decision function  $y_i$  is given by equation

$$y_i = \text{sgn}(\langle \omega, x_i \rangle + b) \quad (12)$$

$\omega$  is the normal vector of the hyperplane  $h$ ,  $b$  is the bias value of the hyperplane  $h$ . It represents the normal distance between hyperplane and origin of the coordinate system.

<sup>4</sup> Source: [http://upload.wikimedia.org/wikipedia/commons/3/39/Random\\_walk\\_in2D\\_closeup.png](http://upload.wikimedia.org/wikipedia/commons/3/39/Random_walk_in2D_closeup.png)

<sup>5</sup> Source: [Mar03] p.8



**Figure 3:** Illustration of hyperplane, that divides test data by classification<sup>5</sup>

**Linear data separation:** The hyperplane  $h$  has to be chosen, so that it represents the largest possible separation, so-called margin between the two defined classes (see Figure 4). The normal vector  $\omega$  is derived by analyzing the training points, which are closest to the hyperplane. The vector representation of these data points are called *Support Vectors*. The objective function of this quadratic program [Bas07] is defined as maximize all distances of each Support Vector to the hyperplane  $h$ . This can be written formally:

$$\begin{aligned}
 &\text{minimize in } (\omega, b) \\
 &\quad \frac{1}{2} \|\omega\|^2 \\
 &\text{subject to} \\
 &\quad y_i(\langle \omega, x_i \rangle + b) \geq 1 \quad \forall 1 \leq i \leq m
 \end{aligned} \tag{13}$$

**Non-Linear data separation:** In most cases, it is not possible to classify data sets with a linear function. This is caused by measurement errors, or nonlinear class structures. In order to perform a linear separation anyway, one has to introduce slack variables  $\xi_i$ . If using slack variables, it will be possible to violate constraints, as long as the sum of all slack variables is sufficiently small enough.

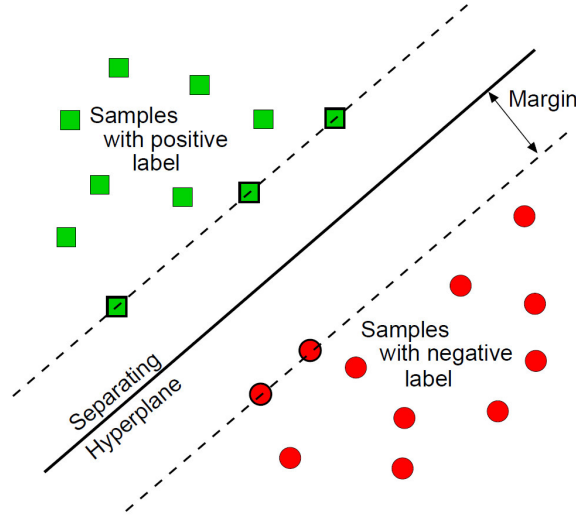
$$\text{minimize } \sum_{i=1}^m \xi_i, \xi_i \geq 0 \tag{14}$$

In order to minimize the sum of slack variables, this sum is added to the objective function of the quadratic program.

This can be written formally:

$$\begin{aligned}
 &\text{minimize in } (\omega, b) \\
 &\quad \frac{1}{2} \|\omega\|^2 + C \sum_{i=1}^m \xi_i \\
 &\text{subject to} \\
 &\quad y_i(\langle \omega, x_i \rangle + b) \geq 1 - \xi_i \quad \forall 1 \leq i \leq m
 \end{aligned} \tag{15}$$

<sup>6</sup> Source: [Mar03] p.12



**Figure 4:** Illustration of maximum-margin hyperplane<sup>6</sup>

These kind of problems are often solved in their dual form. The standard dualization method are the Lagrange multipliers. The normal vector  $\omega$  of hyperplane  $h$  can be written as:

$$\omega = \sum_{i=1}^m \alpha_i y_i x_i \quad (16)$$

Equation (16) implies, that the normal vector  $\omega$  can be a linear combination of training point vectors, the so-called Karush-Kuhn-Tucker condition [Bas07]. All training points  $x_i$  with  $\alpha_i > 0$  and  $y_i(\omega \cdot x_i - b) = 1$  are called Support Vectors. If  $\xi_i > 0$  these support vectors are within the margin range.

Consequently, one can formulate the optimization problem to:

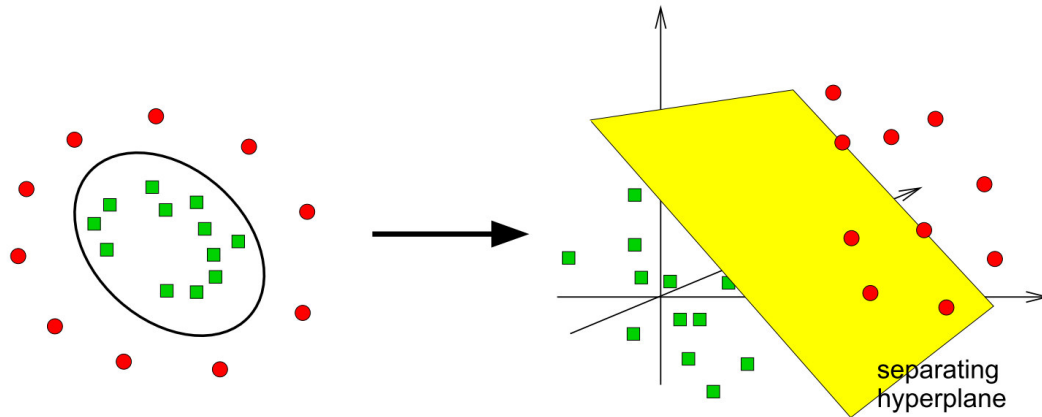
$$\begin{aligned} &\text{maximize in } (\alpha_i) \\ &\sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j x_i^T x_j \\ &\sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j k(x_i, x_j) \end{aligned} \quad (17)$$

with kernel  $k(x_i, x_j) = x_i \cdot x_j$

**Kernel Trick:** Most of data sets can not be separated linearly in a sufficient way. Therefore it is possible to define a hyperplane with a higher dimension, than the data set itself. Figure 5 shows a red and green data set. On the left side, the separating plane is rather complex and nonlinear. After transformation in a higher dimension (right side), it has been possible to define a linear hyperplane, that separates the two data sets perfectly. Therefore, the training point vectors are mapped to a feature space  $J$ , by  $\phi : \mathbb{R}^p \rightarrow J$ . In order to solve the dot product in  $J$  one introduced kernel functions  $k(p, q) = \langle \phi(p), \phi(q) \rangle$ .

<sup>7</sup> Source: [Mar03] p.24





**Figure 5:** Illustration of determine hyperplane in higher dimensions<sup>7</sup>

B. Boser, I. Guyon and V. Vapnik suggested in 1992 to apply a "kernel trick" in order to derive the maximum-margin hyperplane [Bas07, Mar03]. So, it is possible to replace every dot product, which has to be solved in higher dimensions, by a nonlinear kernel function. Kernel function can be:

$$k(x_i, x_j) = (x_i \cdot x_j)^d \quad \text{Polynomial homogeneous} \quad (18)$$

$$k(x_i, x_j) = (x_i \cdot x_j + 1)^d \quad \text{Polynomial inhomogeneous} \quad (19)$$

$$k(x_i, x_j) = \exp(-\gamma \|x_i - x_j\|^2) \quad \text{Gaussian radial} \quad (20)$$

with  $\gamma > 0$

### 2.1.3 Evolutionary Algorithms

Evolutionary algorithms are population-based metaheuristic optimization algorithms [Wei09]. Furthermore, evolutionary algorithms are used to search in parallel global feasible and optimal solutions [Cas08].

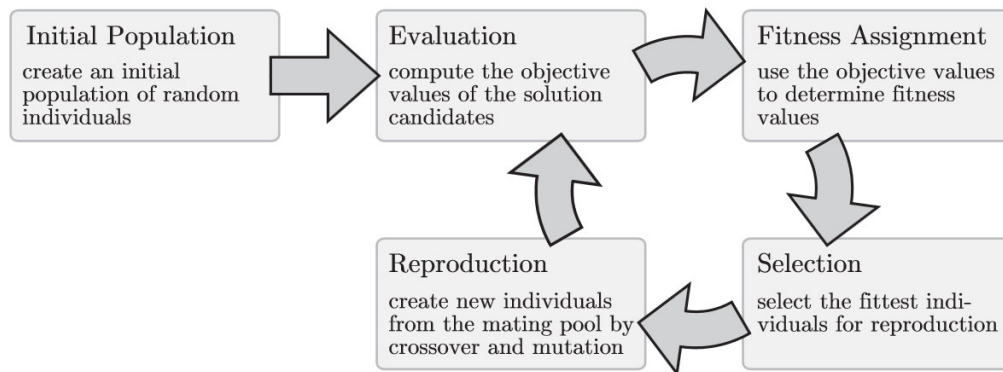
The detection of new solutions, the heuristic, is done by selection and random decisions. The mechanisms are inspired by biological concepts, e.g. mutation, crossover, natural selection and survival of the fittest.

The solutions can be determined iteratively. Another advantage of evolutionary algorithms is caused by their "black box" character. Thus, one does not need many assumptions about the underlying objective. So, the logical system does not have to be understood perfectly.

In 1859, Charles Darwin published "On the Origin of Species" [Dar59]. So, he is the first, who identified the major driving forces behind biological evolution, *natural selection* and *survival of the fittest* [Wei09].

The modern evolutionary algorithms abstract the detected natural behavior, in order to solve NP-hard and NP-complete problems. A set of feasible solutions  $\mathbb{G}$  is given by the set of all possible DNA strings. Its elements  $g \in \mathbb{G}$  are called natural genotypes.

Each individual, or *phenotype*  $x \in \mathbb{X}$  in problem space  $\mathbb{X}$  result from it's DNA configuration.



**Figure 6:** General cycle of evolutionary algorithms<sup>8</sup>

A fitness value is assigned to each phenotype  $x \in \mathbb{X}$ , which reflects the objective function of the problem.

A typical evolutionary algorithm consists of six steps [Wei09]:

1. First, a random initial population  $Pop$  of individuals  $p$  have to be created. Each individual  $p$  is defined by it's specific random genome  $p.g$
2. Each individual have to be evaluated, so the objective function  $f \in F$  of each individual  $p$  have to be simulated or calculated.
3. Afterwards, each individual  $p$  receives it's specific fitness value  $v(p.x)$  by comparison to all other individuals.
4. All individuals  $p$  are classified by it's fitness values. Bad fitness values cause a low probability of reproduction. Individuals with good fitness values enter the mating pool with a higher probability of reproduction.
5. During the reproduction phase, new individuals are created by varying or combining the genotypes  $p.g$  of the selected parent individuals. Afterwards, the new offspring are integrated in the general population.
6. Finally, the optimization meets the termination criterion, so the evolution stops. Otherwise, the algorithm return to step 2.

Evolutionary algorithms are divided into four sub-classes, evolutionary programming, evolutionary strategies, genetic algorithms and genetic programming. Final application in evolutionary algorithms are typically a combination of several sub-classes.

The most important sub-classes are genetic algorithms and genetic programming.

Genetic algorithms are the most popular evolutionary algorithms. John Holland started the studies in this topic in 1960 [Wei09]. Genetic algorithms define several genotypes, which are decoded and quantified. So genetic algorithms are applied in *Genetic Searches*. These kind of algorithms are advantageous to search new solutions, but are less optimal for optimizing problems.

The second important sub-class is genetic programming. The main difference to all other

<sup>8</sup> Source: [Wei09] Figure 2.1

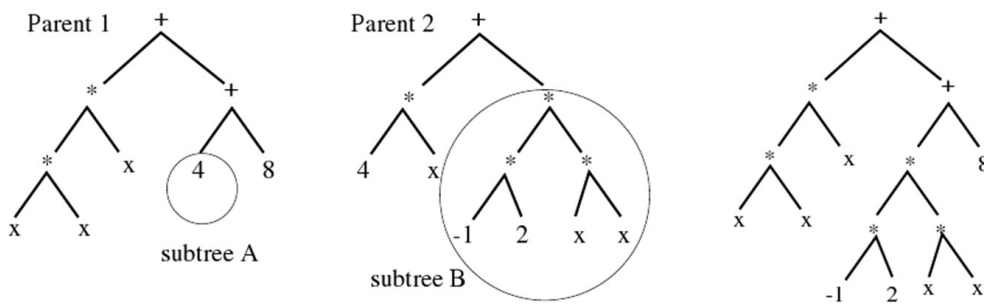
evolutionary algorithms is the fact, that these algorithms do not optimize parameters in order to solve the optimization problem. Genetic programming do automatic programming in order to solve problems.

An example is given by the approximation of analytic functions by genetic programming. These algorithms form programs, e.g. Lisp-programs, so that the deviation of the analytic function becomes minimal.

In order to archive automatic programming, one has to meet several boundary conditions:

- the whole algorithm has to be formulated as tree structure
- this tree structure has to consists of a sub-tree structure, which still deliver syntactical correct programs, after sub-tree parts have been replaced or swapped.
- Each expression consists of functions as inner nodes, e.g. mathematical operations
- the leaves of the tree structure is given by variables and constants

A typical genetic program structure is given in Figure 7.



**Figure 7:** General structure of genetic programs<sup>9</sup>

#### 2.1.4 Linear Integer Programming

In general, optimization deals with the determination of the minimum or the maximum of a given function  $f$  over the points of set  $\chi$ . A continuous function over a compact set attains it's maximum and it's minimum [Bor10]. So, this theorem proofs the existence of global optima, but it does not show how to compute practically the location of the optima. This is the major purpose of optimization.

A *problem description*  $\Pi$  is a pair  $(\iota, S)$  where  $\iota$  contains all instances of  $\Pi$  and for all  $x \in \iota$  the set  $S(x)$  is the set of feasible solutions [Nie09]. Furthermore, the set of feasible solutions  $S$  is defined by a constraint function  $g : \mathbb{R}^n \rightarrow \mathbb{R}^m$ . Thus, we can define  $S = \{x \in \mathbb{R}^n : g(x) \leq 0\}$ . An example for a very simple problem description is a linear equation system: The instance is given by a matrix  $A \in \mathbb{R}^{m \times n}$  and a vector  $b \in \mathbb{R}^m$ . The set of feasible solutions is defined as the set of all  $x \in \mathbb{R}^n$ , so that  $Ax = b$ . The problem is to find a feasible solution, which can be done e.g. by the Gaussian elimination method.

*Optimization problems* are problem descriptions (see [Nie09]) in which its sets of feasible solutions are additionally quantified by objective functions. For all instances  $x \in \iota$  with a

<sup>9</sup> Source: [Cas08] Figure 7

feasible solution  $y \in S(x)$  there exists an objective function value  $f(x, y) \in \mathbb{R}$  [Nie09]. This function quantifies each solution of  $x$ .

Let  $\Pi = (\iota, S)$  be a problem description and for each pair  $x \in \iota$  and  $y \in S(x)$  an objective function value  $f(x, y) \in \mathbb{R}$  exists [Nie09]. So, the problem description  $\Pi^{max}$  is called *maximization problem*, if an instance  $x \in \iota$  exists with a solution  $\bar{y} \in S(x)$  so that  $f(x, \bar{y}) \geq f(x, y)$  for all  $y \in S(x)$ . If this solution does not exist, proof that  $S(x)$  is empty or infinite.  $\bar{y}$  is the optimal solution for  $\Pi^{max}$ .

A general definition for linear optimization problems is given by a function  $f(x) : \mathbb{R}^n \rightarrow \mathbb{R}$  where  $x \in S$ .  $S$  is subset of  $\mathbb{R}^n$ . The maximization or minimization of  $f(x)$  is constrained by function  $g : \mathbb{R}^n \rightarrow \mathbb{R}^m$  with  $S = \{x \in \mathbb{R}^n : g(x) \leq 0\}$ . A maximization problem is a *linear optimization problem* or linear program (LP) if both the objective function and the constraint functions are linear functions from  $\mathbb{R}^n$  to  $\mathbb{R}$  [Bor10]. A linear optimization problem can be written as:

$$\max c^T x \text{ s.t. } Ax = b, x \geq 0 \quad (21)$$

with  $c \in \mathbb{R}^n$ ,  $b \in \mathbb{R}^m$ ,  $A \in \mathbb{R}^{m \times n}$ ,  $x \in \mathbb{R}^n$ .

If the solution vector of a linear optimization problem is fully or partly restricted to integer values, one calls this kind of problems *mixed integer linear program* (MIP). Formally, a MIP is defined by an objective function  $c^T x$ , an constraint system  $Ax \leq b$  and with feasible solutions  $x \in \mathbb{Z}^p \times \mathbb{R}^{n-p}$  where  $c \in \mathbb{R}^n$ ,  $A \in \mathbb{R}^{m \times n}$ ,  $b \in \mathbb{R}^m$  and  $p \in \{0, 1, \dots, n\}$  [Pfe12]. The definition of  $p$  and  $x$  precipitates special cases of MIPs. This definition describes a linear problem if  $p = 0$ . If  $p = n$  all entries in the solution vector have to be integral. This kind of problem is called integer linear program (IP). A further special case of an integer linear program is the binary linear program, with  $p = n$  and  $x \in \{0, 1\}$  [Pfe12].

Integer problems are very common in real life applications. The field of operations research studies the optimization of production flows and logistics. In this work, we apply an integer program with binary variables, in order to derive global optima on discrete data points (see Subsection 4.4.3).

#### 2.1.4.1 Solving Integer Programs

The 0-1 IP problems, i.e. integer programs with binary variables, which are used in this work, are in the set of NP-complete problems (see [Eri12]).

A typical way to solve integer problems, as introduced in this passage, is the *relaxation* of constraints. In this special case, we relax the integer constraints of each variable. The set of feasible solutions of an integer linear program with  $\{x \in \mathbb{Z}^n \mid Ax = b\}$  can be relaxed by a rational  $A \in \mathbb{Q}^{m \times n}$ ,  $b \in \mathbb{Q}^n$  polyhedron  $P := \{x \in \mathbb{R}^n \mid Ax \leq b\}$ . A rational polyhedron  $P := \text{conv}\{x \in \mathbb{Z}^n \mid Ax = b\}$  includes the integral hull  $P_I$ , i.e. the convex hull of the integer solution is completely inside  $P$

$$P_I \subseteq P. \quad (22)$$

Both polyhedra are illustrated in Figure 8. The blue polyhedron  $P$  is defined by the linear constraints:

$$\begin{aligned} -x + y &\leq 1 \\ 2x + 3y &\leq 12 \\ 3x + 2y &\leq 12. \end{aligned} \quad (23)$$

If  $y$  has to be maximized, the optimal value of the LP would be  $(1.8, 2.8)$  with value 2.8. The red set of points are the valid vectors for the integer program. The dashed line shows the boundary of the convex hull of  $P_I$ . The optimal solutions for the integer program are  $(1, 2)$  and  $(2, 2)$  with value 2.

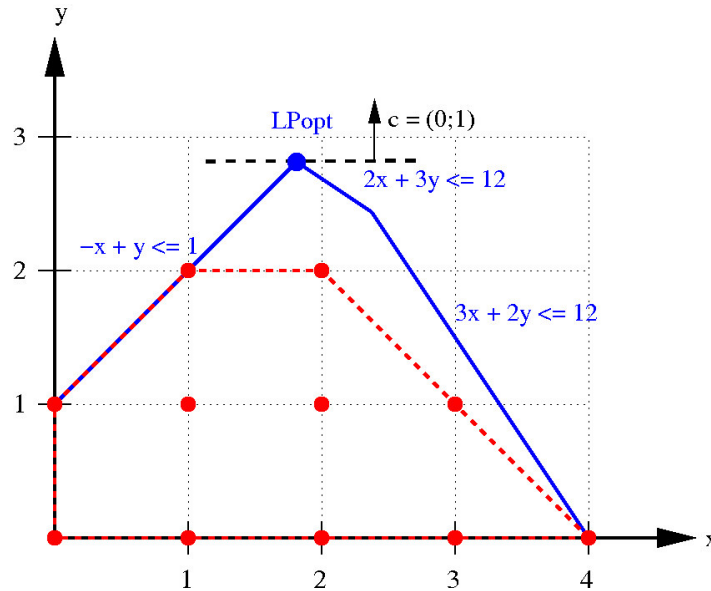


Figure 8: IP polytope with LP relaxation<sup>10</sup>

There exist several algorithms, which solve integer programs. A complete enumeration of all feasible solutions is not applicable for more complex problems. In the worst case one has to do the complete enumeration, but if possible, one should avoid this. Therefore, mathematicians have developed several algorithms, that solve practical instances of NP-hard and NP-complete problems.

One differentiates between exact algorithms and heuristics. Most exact algorithms have exponential run-time, efficient exact algorithms can have polynomial runtime. Heuristic approaches are often customized for the specific problem, so the run-time is less than the one of exact algorithms. However, a common problem of heuristic approaches is the detection of global optima. It is possible, that heuristics detect local optima, only. Many heuristic approaches derive solutions by iterations.

The following paragraphs, introduce two exact algorithms, that solve NP-hard problems, the *Cutting Plane* and the *Branch and Bound* method.

**Cutting Plane:** The cutting plane algorithm was applied for the first time in the work of Gomory et al. in 1960 [Mar02].

The definition and discussion of the following algorithm has been taken from M. Pfetsch [Pfe12] and H. Marchand et al. [Mar02]. The algorithm consists of the following two steps:

1. Solve the relaxed linear program of the IP with a suitable algorithm, e.g. Simplex-algorithm.

<sup>10</sup> Source: [http://upload.wikimedia.org/wikipedia/commons/thumb/f/fd/IP\\_polytope\\_with\\_LP\\_relaxation.png/350px-IP\\_polytope\\_with\\_LP\\_relaxation.png](http://upload.wikimedia.org/wikipedia/commons/thumb/f/fd/IP_polytope_with_LP_relaxation.png/350px-IP_polytope_with_LP_relaxation.png)

2. If the derived solution consists a non-integer vector, one has to introduce new constraints, that

- a) make the found solution infeasible, by cutting off the current solution of the LP
- b) do not cut off any integer solution of the integer program

We will show at the example of Gomory fractional cuts how such cutting planes can be derived. Let an integer problem be given by

$$\begin{aligned} \max \quad & c^T x \\ \text{Ax} = & b \\ x \geq & 0 \\ x \in & \mathbb{Z}^n \end{aligned}$$

with a corresponding polyhedron  $P = \{x \in \mathbb{R}^n : Ax = b, x \geq 0\}$  and a second polyhedron  $P_I = \text{conv} \{x \in \mathbb{Z}^n : Ax = b, x \geq 0\}$ , which is given by the convex hull of the set of feasible integer solutions.

Equation (22) shows that the polyhedron  $P$  includes the restricted polyhedron  $P_I$ . Furthermore, we assume an optimal and feasible solution in  $P_I$ . The general idea of the cutting plane algorithm is to add further constraints to  $P$ , in order to obtain an integer optimal solution for Equation (24). We assume, that  $A$  and  $b$  are integral.

Corresponding to step 1, one has to solve the IP without integer constraint, with e.g. the simplex algorithm. So, we obtain an optimal solution  $x^*$  and an optimal basis  $B$  with  $B \subseteq \{1, \dots, n\}$ ,  $|B| = m$  and  $A_B$  is regular. The optimal solution of the relaxed problem  $x^* = A_B x_B^* + A_N x_N^* = b$  can be expressed with the new basis  $B$ :

$$\begin{aligned} x_B^* &= A_B^{-1} b - A_B^{-1} A_N x_N^* \\ x_N^* &= 0 \end{aligned}$$

If  $x^*$  is already integral, the algorithm terminates with the optimal integer solution. Otherwise, one of the values in  $x_B^*$  must be fractional. Let  $i \in B$  be an index with  $x_i^* \notin \mathbb{Z}$ . Since all feasible integer solutions fulfill Equation (24) we obtain:

$$x_i^* = A_{i,\cdot}^{-1} b - \sum_{j \in N} A_{i,\cdot}^{-1} A_{\cdot,j} x_j \in \mathbb{Z} \quad (24)$$

Equation (24) will remain integer, if one adds integer multiplies of  $x_j, j \in N$  or an integer value to  $A_{i,\cdot}^{-1} b$ . So, we obtain:

$$f(A_{i,\cdot}^{-1} b) - \sum_{j \in N} f(A_{i,\cdot}^{-1} A_{\cdot,j}) x_j \in \mathbb{Z}_0^- \quad (25)$$

for all integral solutions  $x$  with the function  $f(\alpha) = \alpha - \lfloor \alpha \rfloor$  for  $\alpha \in \mathbb{R}$ . Since  $0 \leq f(\cdot) < 1$  and  $x \geq 0$ , we obtain:

$$f(A_{i,\cdot}^{-1} b) - \sum_{j \in N} f(A_{i,\cdot}^{-1} A_{\cdot,j}) x_j \leq 0 \quad (26)$$

which is equivalent to

$$\sum_{j \in N} f(A_{i,\cdot}^{-1} A_{\cdot,j}) x_j \geq f(A_{i,\cdot}^{-1} b) > 0 \quad (27)$$

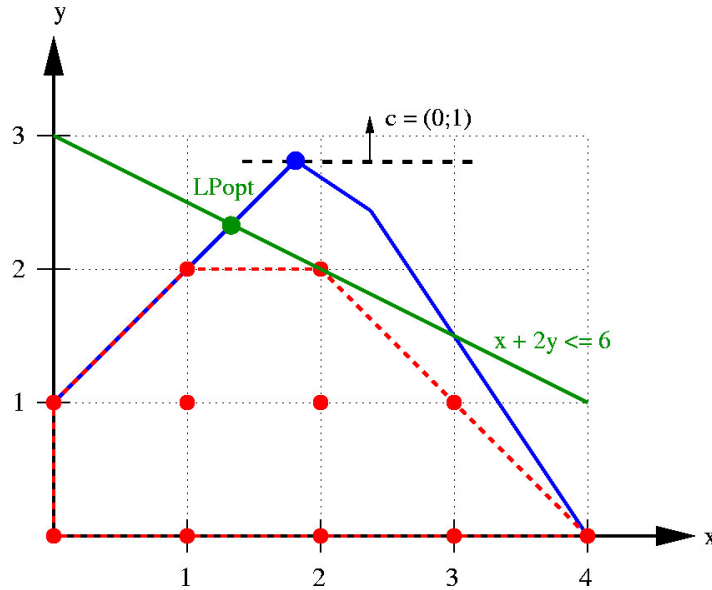
Moreover  $x^*$  violates this equation, because  $x_N^* = 0$  and  $f(A_{i,\cdot}^{-1} b) = f(x_i^*) > 0$ .

Equation (27) defines a hyperplane which cuts  $P$ , furthermore this equation contains integer values on the left side. This can be shown by subtracting  $x_i + \sum_{j \in N} A_{i,\cdot}^{-1} A_{\cdot,j} x_j = A_{i,\cdot}^{-1} b$  from Equation (27). So, we obtain:

$$x_i + \sum_{j \in N} \lfloor A_{i,\cdot}^{-1} A_{\cdot,j} \rfloor x_j \leq \lfloor A_{i,\cdot}^{-1} b \rfloor \quad (28)$$

This inequality can be added to the instance  $Ax = b$  and all values remain integral. Moreover, the newly introduced slack variables can be required to be integer. Thus the procedure can be iterated.

The basic idea of the cutting planes algorithm is illustrated in Figure 9. The green cutting plane  $LP_{opt}$  is the final cut, that lead to the optimal solution of the integer program.



**Figure 9:** Basic idea of cutting planes algorithm<sup>11</sup>

The *Cutting plane* algorithms is not applied directly in real-world problems. More complex problems cause many additional constraints, so the optimization process becomes numerically instable. This method is combined with the *Branch and Bound* algorithm, which will be explained in the next paragraph.

<sup>11</sup> Source: [http://commons.wikimedia.org/wiki/File:Cutting\\_plane\\_algorithm.png](http://commons.wikimedia.org/wiki/File:Cutting_plane_algorithm.png)

**Branch and Bound:** The *Branch and Bound* algorithm is a method that tries to avoid the full enumeration of all solutions of an optimization problem [Kor08]. This algorithm has been introduced by Land and Doig in 1960 and has been applied for the first time by Little et al. in 1963 in order to solve the traveling salesman problem [Kor08]. This exact algorithm excludes less optimal solutions by the determination of lower and upper bounds. So, the algorithm is able to exclude sub-sets of the set of feasible solutions. In this work we apply this exact method in order to solve mixed integer problems. This algorithm will have exponential runtime [Pfe12].

Basically, the algorithm is divided into two steps: the branching and the bounding. In the first step, the algorithm branches the problem into two non-empty sub-sets. Afterwards, the algorithm determines a upper bound on the objective function of both solution sub-sets. So, the algorithm is able to exclude branches, which do not contain better solutions for the optimization problem [Kor08]. The algorithm solves a mixed integer problem as introduced in Equation (24). In the first step, the algorithm computes an upper bound of the problem by solving the LP-relaxation of the problem, so the problem is solved without the restriction of integral values. If the computed solution is not integer, the algorithm has to subdivide the problem into sub problems, which are computed recursively. So, the Branch and Bound algorithm creates a sequence of sub-problems with reduced scopes. The branching method is equivalent to the creation of a sequence of pairwise disjoint polyhedra in  $\mathbb{R}^n$  [Nie09]. Consequently, the optimal solution of the MIP is within one of the created polyhedra, so the algorithm does not lose any optimal feasible solution.

The branch and bound method is part of the most modern mixed integer problem solvers, since the algorithm solves a variety of practical instances of optimization problems with better runtimes than exponential ones.

The resulting algorithm of the branch and bound method is given by four steps [Kor08]:

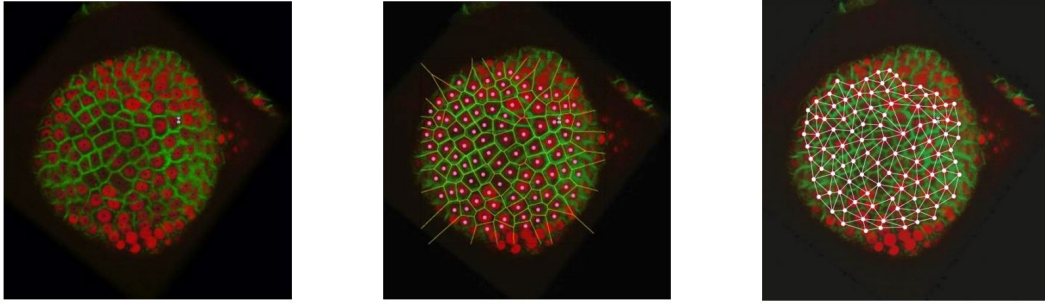
- Define the original data tree  $T := (\{S\}, \emptyset)$ , with the set of feasible solutions  $S$ .  
Afterwards, marks  $S$  as active.  
Define the lower bound  $L := \infty$ . Alternatively, one can apply a heuristic method in order to obtain a better value for the lower bound.
- Choose an active node  $X$  of the tree  $T$ . If this is not possible, the algorithm will terminate (**STOP**).  
Mark  $X$  as inactive.  
Now, the algorithm performs the branching. Determine a partition  $X = X_1 \dot{\cup} \dots \dot{\cup} X_t$ .
- **For** each  $i = 1, \dots, t$  **do**:  
    Now, the algorithm performs the bounding. So, it determines the upper bound  $U$  of the costs of all feasible solutions in  $X_i$ .  
    **If**  $|X_i| = 1$  and  $\text{cost}(S) > L$  **then**:  
        Set  $L := \text{cost}(S)$  and  $S^* := S$ .  
    **If**  $|X_i| > 1$  and  $L < U$  **then**:  
        Set  $T := (V(T) \cup \{X_i\}, E(T) \cup \{\{X, X_i\}\})$  and mark  $X_i$  as active. So the tree  $T$  is expanded by an additional node  $(V(T) \cup \{X_i\})$  and by an additional edge  $(E(T) \cup \{\{X, X_i\}\})$ .
- **Go to** step 2.



### 2.1.5 Voronoi Diagrams

Data point distributions can be structured into regions or sites. One method is the computation of Voronoi Diagrams, the Voronoi tessellation [Ott99]. A general definition of Voronoi sites of a point  $p$  is the set of all points, which are closer to  $p$  than to any other point. An example in nature is given in Figure 10. Every region contains points, that are closer to it's center than to any other center data point of other regions. These kind of regions are called *Voronoi cells*. The set union of all *Voronoi cells* is called *Voronoi Diagram*. The dual graph of a *Voronoi Diagram* is called *Delaunay Triangulation* [Boi10] (see Figure 12).

To be more precise, let  $S$  be a finite number of *Sites* (eq. 29).



**Figure 10:** Example for Voronoi Tessellation in nature<sup>12</sup>

$$S \subset \mathbb{R}^n \quad (29)$$

Furthermore, let a distance function  $\text{dist}(x, y)$  be defined in  $\mathbb{R}^n$  with:

$$\text{dist}(x, y) := \|x - y\| = \sqrt{(x - y)^T (x - y)} \quad (30)$$

*Voronoi Regions*  $VR_S(s)$  are defined by

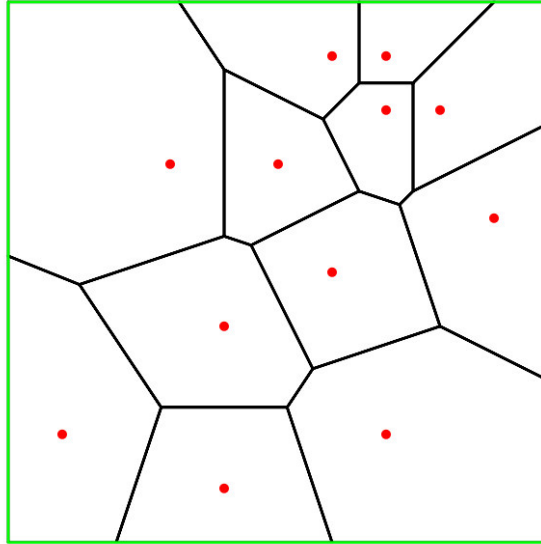
$$VR_S(s) := \{x \in \mathbb{R}^n | \text{dist}(x, s) \leq \text{dist}(x, q) \forall q \in S\} \quad (31)$$

$$s \in S$$

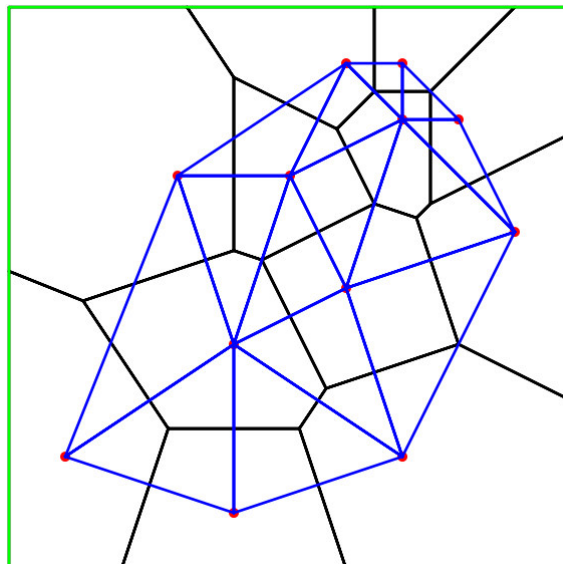
$x \in VR_S(s)$  means, that  $s$  is the nearest site of  $S$  of point  $x$ . Figure 11 show the Voronoi Diagram (black) of a set of points (red).

Historically, Descartes used informally Voronoi diagrams in 1644 (*Principia Philosophiae*). Peter Gustav Lejeune Dirichlet used two and three dimensional diagrams in order to study quadratic forms in 1850. Voronoi tessellation are named after Georgy Fedosierych Voronyi in 1908.

Voronoi diagrams are also known as Thiessen- Polygons and Dirichlet tessellation.



**Figure 11:** Illustration of a Voronoi Diagram.<sup>13</sup>



**Figure 12:** Illustration of the dual graph, Delaunay triangulation (blue).<sup>14</sup>



**Figure 13:** Set of point  $P$  (left) and convex hull  $\text{conv}(P)$  of the set of points  $P$  (right)<sup>15</sup>

Another way to determine a Voronoi diagram is by computing a convex hull. A convex hull  $\text{conv}(P)$  of a set of points  $P$  is defined as smallest convex set, that contains all points (see Figure 13).

$$\text{conv}(X) = \left\{ \sum_{i=1}^n \alpha_i x_i \mid x_i \in X, n \in \mathbb{N}, \sum_{i=1}^n \alpha_i = 1, \alpha_i \geq 0 \right\} \quad (32)$$

The determination of convex hulls is a fundamental challenge in algorithmic geometry. Many problems can be reduced to the calculation of convex hulls, e.g.:

- Delaunay triangulations and Voronoi tessellation
- power diagrams
- halfspace intersections

A Delaunay triangulation in  $\mathbb{R}^d$  can be derived by computing a convex hull in  $\mathbb{R}^{d+1}$ . So, the set of points in  $\mathbb{R}^d$  have to be lifted on a paraboloid in  $\mathbb{R}^{d+1}$ . So, one adds an additional component to all data point vectors  $d_i = (d_1, \dots, d_d)$ . This additional component is given by the norm of the vector  $\|d\|$ .

Typically, it is easier to solve a convex hull, than a Voronoi Tessellation directly. Common algorithms, to determine the convex hull of a set of points are cdd+ by Komei Fukuda [Fuk04] and beneath-and-beyond [Jos02]. The beneath-and-beyond algorithm will be discussed in Subsection 2.3.2.

---

### 2.1.6 Linear Regression

---

A common numerical problem in science is the approximation of numerical data to an analytic function. By describing data points with analytic functions, it is possible to do further data analysis, e.g. interpolation and extrapolation or comparison to model behavior. This process is commonly called "fitting". The linear regression method is a simple special case of data fitting.

The model value  $y_i$  is given by a simple linear equation

$$y_i := a + bx_i + \epsilon_i = \hat{y}_i + \epsilon_i$$

---

<sup>12</sup> Source: [Boi10], p. 45

<sup>13</sup> Source: [Gal12], p. 149, Figure 8.2

<sup>14</sup> Source: [Gal12], p. 152, Figure 8.4

<sup>15</sup> Source: [Boi10], p. 2

with  $\epsilon_i = y_i - \hat{y}_i$ .  $\epsilon_i$  describes the gap between model and measured value ([Sac09] cf. eq. 3.74).

The goal of the linear regression method is to choose  $a$  and  $b$ , so that the sum of the squared deviations from the monitored  $y$ -values and the evaluated regression function becomes minimal. The deviations  $\epsilon$  are called residuals [Sac09].

The ordinary least-squares method (OLS) can be solved by equations 33 and 34 ([Sac09] cf. eq. 3.75).

$$b = \frac{n \sum x_i y_i - \sum x_i \sum y_i}{n \sum x_i^2 - (\sum x_i)^2} = \frac{S_{xy}}{S_x^2} \quad (33)$$

$$a = \bar{y} - b\bar{x} \quad (34)$$

Historically, Francis Galton named this method during his studies of body sizes of parents and children under the aspect of heredity [Sac09]. He defined two principal confinements for the fitting process. The first condition is that the variation of  $x$  should be sufficiently large, in order to increase the quality of the prognosis. The second one is that the difference  $d_i = (y_{i+1} - y_i)$  and the second difference  $(d_{i+1} - d_i)$  do not contain a trend. Formally, the second deviation should be close to zero [Sac09].

---

### 2.1.7 Shortest Path Problem

---

One of the most common problems in the topic of combinatorial optimization is the shortest path problem [Kor08]. The following definition is taken from Hlineny and Moris [Hli12] and Korte and Vygen [Kor08].

The definition of the shortest path problem is given by finding the shortest path between two given nodes of a directed graph  $G$ . The directed graph  $G$  is a pair of a finite set  $V(G)$  of vertices and a finite multi set  $E(G) \subseteq V(G) \times V(G)$  of edges.  $G$  has defined weights  $c : E(G) \rightarrow \mathbb{R}$  and two nodes  $s, t \in V(G)$ . Finding the shortest path  $P$  between node  $s$  and  $t$  means that  $P$  contains the minimal weight  $c(E(P))$  or no way exists between  $s$  to  $t$  [Kor08, Hli12].

These kind of route planning processes can be separated into *static* and *dynamic* processes. *Static* planning implies, that the network graph  $G$  is fixed during computation. The *dynamic* process allows the network to change in time, be effected by road constructions or traffic jams.

The principle of determining the shortest path in a directed graph is applied in this work. In order to improve the measurement process, this principle is integrated into the *Test Drive* module. This module will be discussed in Subsection 4.2.3. This problem for directed graphs  $G$  with non-negative weights  $c$  is solved with the Dijkstra algorithm. This algorithm will be discussed in the next paragraph.

---

### 2.1.8 Dijkstra Algorithm

---

One algorithm to compute the shortest path from one vertex to any other vertices in a graph is the Dijkstra Algorithm. The following algorithm description has been taken and translated from Korte and Vygen ([Kor08] pp 168-169).

This algorithm determines the shortest path  $P$  on a directed graph  $G$  with weights  $c : E(G) \rightarrow \mathbb{R}_+$  (Definition given in 2.1.7) between the starting vertex  $s$  and the destination vertex  $t \in V(G)$ .

The algorithm returns the shortest paths from  $s$  to all vertices  $v \in V(G)$  and its lengths  $l(s)$ .  $l(s)$  is the length of the shortest path between  $s$  and  $v$ , furthermore, the algorithm computes the direct predecessor of  $v$  on the shortest path  $p(v)$ . So the shortest path is given from  $s$  to  $p(v)$  and the edge between  $p(v)$  and  $v$ .

The algorithm in detail is given by:

1. Set  $l(s) := 0$ . Set  $l(v) := \infty \forall v \in V(G) \setminus \{s\}$ .  
Set  $R := \emptyset$   $R$  is the set of chosen vertices.
2. Choose vertex  $v \in V(G) \setminus R$  with  $l(v) = \min_{w \in V(G) \setminus R} l(w)$ .
3. Set  $R := R \cup \{v\}$
4. **For** all  $w \in V(G) \setminus R$  with  $(v, w) \in E(G)$  **do**:  
    **If**  $l(w) > l(v) + c((v, w))$  **then**  
        set  $l(w) := l(v) + c((v, w))$  and  $p(w) := v$
5. **If**  $R \neq V(G)$  **then go to** step 2

The runtime of the Dijkstra algorithm of a Graph  $G$  with  $n$  vertices and  $m$  edges is given in Equation (35) (see [Kru11] p 5).

$$\mathcal{O}(m + n^2) \quad (35)$$

In order to speed up the algorithm, the Dijkstra algorithm can be expanded with *Binary Heaps* (see 2.3.1). So the runtime improves to (see [Kru11] p 7)

$$\mathcal{O}((n + m)\log(n)) \quad (36)$$

The applied algorithm in this work for determining the shortest path in a directed  $n$ -dimensional graph has been supplemented with priority queues. These priority queues will be discussed in Subsection 2.3.1.

---

## 2.2 Engineering

---

This subsection introduces and explains relevant topics out of the field of engineering. In particular, this passage introduces important actuators and sensors of the *Virtual Test Bench* VTB (see Subsection 4.2.6) and the design of experiment model contributed by AVL [AVL13]. In the next part of this subsection, we give a short introduction into emission standards and quantification cycles. Finally, the difference in measurement methods is studied. This passage is focused on stationary, quasi-stationary and transient measurements.

---

### 2.2.1 Actuators

---

A very basic setup of a spark-ignited engine is illustrated in Figure 14. The actuators are introduced shortly in the following passage. The description of the following components has been taken from K. Grote [Gro11], H. Grohe et al. [Gro10] and RFH [RFH07].

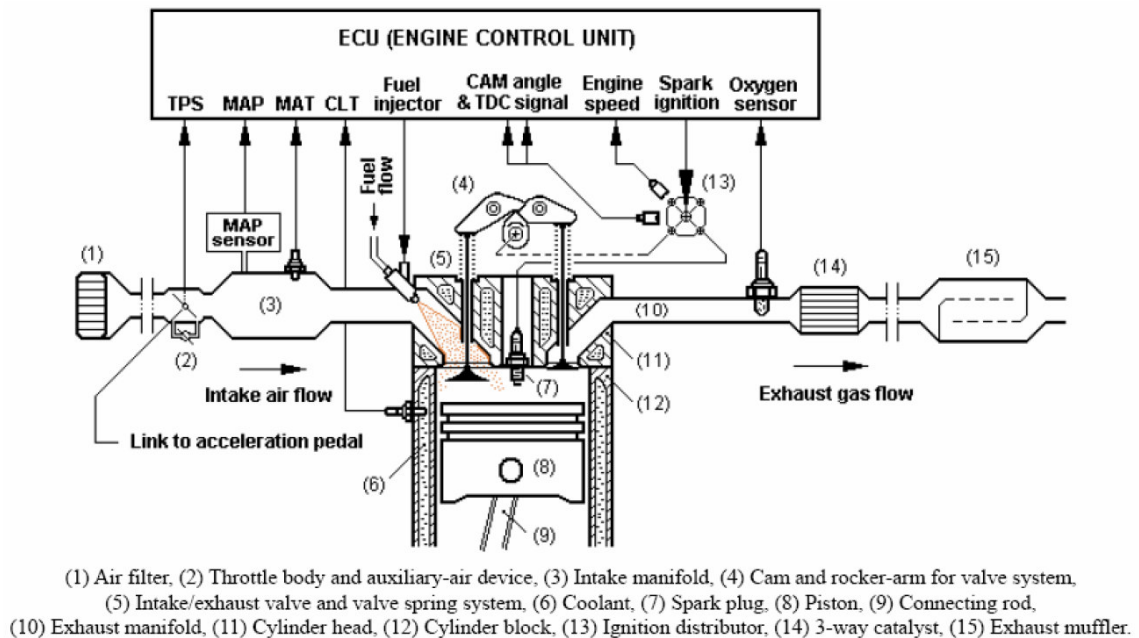


Figure 14: General setup of spark-ignited engines<sup>16</sup>

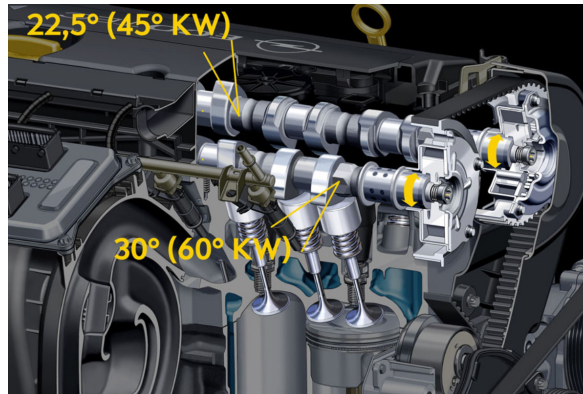
#### 2.2.1.1 Actuators of Virtual Test Bench

**ActDynoSpeed** [ $\frac{1}{min}$ ]: Revolutions per minute of crankshaft.

**CAM\_ex and CAM\_in** [°CA]: The exhaust/intake valves are controlled by the rotation of the exhaust/intake camshaft (CAM\_ex / CAM\_in). Variable Valve Timing VVT has been introduced in order to increase the variability of the engine setup. VVT is able to modify the start/ end of lift, the lift function and the lift amplitude. A current system is shown in Figure 15. By varying the intake/exhaust cross section, one changes the gas cycle of the internal combustion engine (ICE) (see Figure 14 (5)). The phase shift of the camshaft is given in relation to the angle of the crankshaft (°CA). The revolution frequency of the camshaft is the half of the revolution frequency of the crankshaft.

---

<sup>16</sup> Source: [Sit06], Figure 2-1



**Figure 15:** Variable camshaft phasing<sup>17</sup>

**ZW [°CA]:** The spark angle *ZW* is relative to the top dead center TDC of the crankshaft. The ignition plugs cause an electric spark at *ZW*. The ignition delay is the time between *ZW* and the pressure rise in the cylinder. The ignition delay is caused by chemical reactions, which cause the production of chemical radicals, this process finalizes into a chain-reaction. This delay is not constant, but depends on compression ratio, lambda, temperature (see Figure 14 (7)).

**AGR []:** The exhaust gas recirculation system EGR retransfers exhaust gas in the intake channel. So, partial burnt fuel can be burned once more. Typically, the EGR system is realized with a EGR valve and an additional exhaust gas cooler. In this model the AGR input value is given a value between 0 and 1, which determines the flow of residual gas through the EGR valve.

**Alpha []:** The air valve angle *Alpha* controls the amount of fresh air in the intake channel. This air is available for combustion. Depending on the amount of air and the selected value of lambda (air to fuel fraction), the injection system sprays a corresponding mass of fuel (see Figure 14 (2)). In this model *Alpha* is given as value between 0 and 1. It describes the relative flow through the air valve.

### 2.2.1.2 Actuators of AVL Engine Model

This model is based on measurements on a compression ignition/diesel engine.

**N [ $\frac{1}{min}$ ]:** see 'ActDynaSpeed' in Paragraph 2.2.1.1.

**Q [ $\frac{mm^3}{cycle}$ ]:** In contrast to a spark-ignited engine the injected amount of fuel *Q* is the most important actuator. More precisely, the injection process is crucial in the application process of diesel engines. The injection process is given by several pre/pilot-injections, a main injection and post-injections. Typically, the engine torque is mainly given by the main-injection *Q*. This actuator defines the total amount of fuel per cycle. In this simple model, the injected fuel volume is divided into one pilot and the main injection.

**phiMI [°CA]:** The main timing *phiMI* is comparable to the spark timing of Otto-engines. It defines the start timing of chemical reactions in crankshaft angle of the main injection. Similar to the Otto engine, the pressure rise is delayed by the ignition delay.

<sup>17</sup> Source: Adam Opel AG



---

**RailP** [*hPa*]: The pressure in the common rail system is given by *RailP* in hPa. In contrast to solenoid-controlled unit injector elements, the pressure is generated by a central fuel- and high pressure- pump. The fuel injectors are opened and closed by piezo elements.

**MAF** [ $\frac{mg}{hub}$ ]: see 'Alpha' in Paragraph 2.2.1.1. Similar to a spark ignited engine, an air valve controls the amount of air, which contribute to the combustion process. In this model the amount of air is given directly in mass per piston stroke.

**VTG** []: Modern turbochargers do not have a static turbine geometry. Variable-turbine-geometry turbochargers are able to tune the angle of the turbine blades *VTG* in order to increase the amount of boost. Alternative setups are given by static turbines with waste gates. Waste gates are applied to reduce the amount of exhaust gas, that accelerates the turbine, so the amount of boost can be controlled. In this model *VTG* is given as value between 30 and 85.

**qPil and tiPil** [ $\frac{mm^3}{cycle}$ ] [ $\mu s$ ]: The pilot fuel injection is defined by the total amount of fuel *Q*, the absolute part, which contribute to the pilot injection *qPil* and the relative timing *tiPil* to the main timing *phiMI*. In modern car engines, one defines much more than one pilot injection, in order to fulfill emission standards and in order to reduce the noise level of the engine.

---

## 2.2.2 Sensors

---

### 2.2.2.1 Measurands of Virtual Test Bench

The *Virtual Test Bench* system is an empirical approach, but does not predict exhaust emissions.

**b\_e** [ $\frac{kWh}{g}$ ]: The specific fuel consumption *b\_e* is defined by the amount of burnt fuel and the current power level of the engine (see Equation (90)).

**T\_crit** [*K*]: The critical temperature *T\_crit* is defined as temperature of the burnt zone (see eq. 82) at the moment, when the exhaust valves opens. In case of bad timings, the fuel has not been consumed completely, so the exhaust valves would release flames to the exhaust manifold, catalysts and turbocharger. *T\_crit* indicates damages to sensitive parts of the engine setup.

**p\_max** [*bar*]: The maximal point *p\_max* of the cylinder pressure sequence. Every engine is specified on a maximal cylinder pressure, in order to avoid damage on the devices. Typically, the maximal pressure is around 160 bar.

**ActDynoTorque** [*Nm*]: The produced torque of the engine *ActDynoTorque* can be derived by Equation (89). The revolution frequency and the engine torque defines the power level of the ICE.

### 2.2.2.2 Measurands of AVL Engine Model

**MF\_FUEL** [ $\frac{kg}{h}$ ]: The fuel mass flow *MF\_FUEL* displays the amount of fuel, which is delivered to the cylinder per hour.

**CO, HC, NOX and SMKS\_FSN** [*ppm*]: The exhaust emissions are measured in ppm (parts per million). During operation the combustion process generates exhaust emissions. The most important emissions are explained in Subsection 2.2.3.

**IMEP** [*bar*]: The brake mean effective pressure *IMEP* in bar is the average pressure over a cycle in the cylinder.



**LAMBDA []:** Lambda displays the chemical partitioning of the fuel. Spark-ignited engines run on lambda around 1. The combustion limits are 0.6 and 1.6. The lambda value for compression ignition engines is much higher, up to 20. lambda is defined in Equation (38).

$$AFR = \frac{m_{air}}{m_{fuel}} \quad (37)$$

$$\lambda = \frac{AFR}{AFR_{stoich}} \quad (38)$$

$AFR_{stoich}$ : stoichiometric fraction of air to fuel

**TORQUE [Nm]:** see 'ActDynoTorque' in Paragraph 2.2.2.1.

**PSR [bar]:** The manifold pressure  $PSR$  measures the absolute pressure in front of the intake channel. The pressure depends on the absolute environmental pressure and the boost level of the turbocharging system.

**P2 [bar]:** The boost pressure  $P2$  is created by the turbocharging unit. This absolute pressure depend on the  $VTG$  setting and the current combustion behavior.

**PMAX [bar]:** see 'p\_max' in Paragraph 2.2.2.1.

**TSR [°C]:** The manifold temperature  $TSR$  measures the temperature in the intake channel.

**T3 [°CA]:** see 'T\_crit' in Paragraph 2.2.2.1.

**MF\_CO, MF\_HC and MF\_NOX [ $\frac{g}{h}$ ]:** These sensors are similar to  $CO$ ,  $HC$ ,  $NO_x$ . They measure the absolute amount of exhaust emissions per hour.

**be [ $\frac{g}{kWh}$ ]:** see 'b\_e' in Paragraph 2.2.2.1.

---

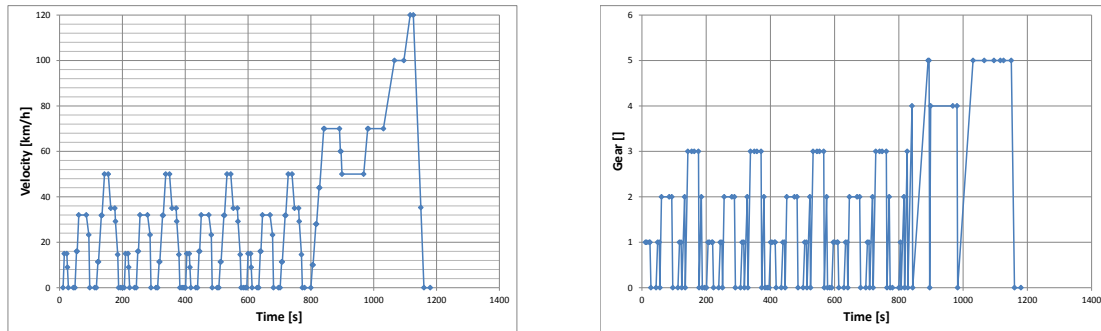
### 2.2.3 Emission Regulation and Quantification Cycles

---

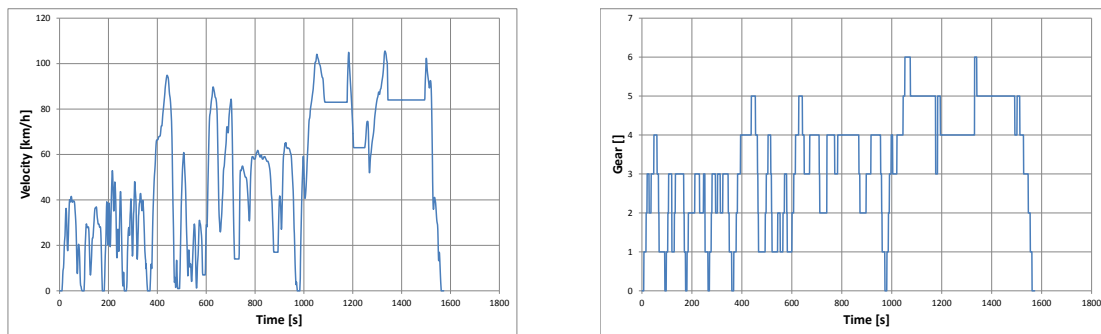
Internal combustion engines consume fossil fuel, e.g. gasoline, diesel or natural gas and transform chemical energy into kinetic energy in order to accelerate the vehicle. But furthermore, the engine transforms the fuel not only into water  $H_2O$  and carbon dioxide  $CO_2$ . The most common exhaust emissions of internal combustion engines are carbon monoxide  $CO$ , hydro carbons  $HC$ , nitrogen oxides  $NO_x$  and soot particles. The maximal emission per distance has been regulated by the European Union (see [EC07]) and its pendants in the US. The regulation defines beside the upper limits for  $CO$ ,  $HC$ ,  $NO_x$  and particles, the test procedure. In the test procedure, the tail-pipe emissions of vehicles are measured after a cold start in a constant environment. The test continues with a standardized test procedure in the laboratory [JRC13]. Current car engines have to pass the "New European Driving Cycle (NEDC)", which consists of the urban cycle ECE-15 and the extra-urban cycle EUDC. The temporal profile is shown in Figure 16. The driving cycle contains four times the ECE-15 cycle and one time the EUDC passage.

The average speed of this test procedure is 34 km/h. In general, this test procedure does not represent a typical driver profile [JRC13]. That is why the European Union define a new test procedure in order to represent real world emissions and driver behavior [Wei13]. The new so called "Random Cycle" is a modal composite of randomly picked path pieces of representative driver patterns [JRC13]. The "Random Cycle" consists of rural-motorway, rural-urban, rural-uphill/downhill and motorway distances.

The "Random Cycle" covers a bigger area of common new car engines' operation points, thus the test procedure covers much more engine operation states. Further, the new test



**Figure 16:** Operational profile of NEDC



**Figure 17:** Operational profile of Random driving cycle.

procedure represents typical driver behavior, so fuel consumption and exhaust emission prediction errors should be much smaller than with the NEDC. The emission standards Euro 5 and Euro 6 are defined in [EC07]. A summary is given in Table 2 and 3.

In order to meet emission limits, the original equipment manufacturer OEM applies after-treatment of exhaust gas systems. These systems have to reduce the absolute amount of critical exhaust species. The first after-treatment system is the oxidation catalyst. Modern after-treatment systems are selective catalytic reduction catalysts, Urea injections and particle filters (see Figure 18). The operational profiles (see Figure 16 and 17) have to be transformed from functions of time ( $v(t), g(t)$ ) to weighting tables  $dt(v, M)$ . The temporal resolution of the operational profiles is one second.

<sup>18</sup> Source: <https://de.wikipedia.org/wiki/Abgasnorm>, [JRC13]

<sup>19</sup> Source: <https://de.wikipedia.org/wiki/Abgasnorm>, [JRC13]

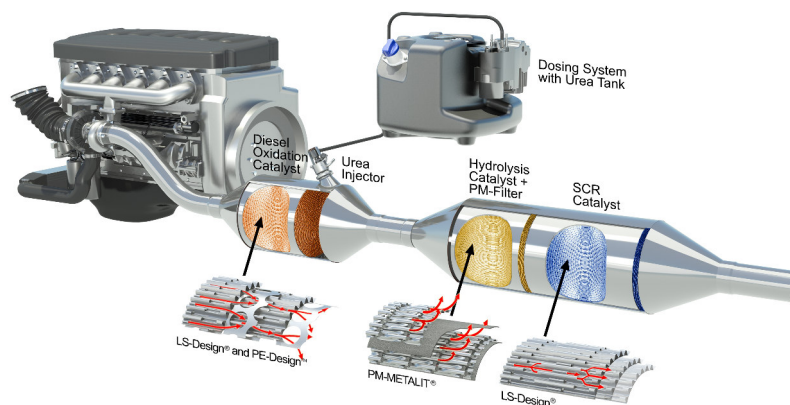
<sup>20</sup> Source: Emitec [http://www.emitec.com/...\\_upload/Presse/Pressefotos/201109\\_IAA\\_SCR\\_System.JPG](http://www.emitec.com/..._upload/Presse/Pressefotos/201109_IAA_SCR_System.JPG)

**Table 2:** Emission limits for passenger cars with spark-ignited engines. Units: mg/km but PN in (1/km)

Norm	Euro 3	D3	Euro 4	D4	Euro 5	Euro 6
Type approval	since 2000		since 2005		since 2009	proposed for 2014
Initial registration	since 2001		since 2006		since 2011	proposed for 2015
CO	2300	1500	1000	700	1000	1000
(HC + NOx)						
NOx	150	170	80	80	60	60
HC	200	140	100	70	100	100
pro rata NMHC					68	68
PM					4,51	4,51
PN	–	–	–	–	–	6E11 /km <sup>18</sup>

**Table 3:** Emission limits for passenger cars with compression ignition engines. Units: mg/km but PN in (1/km)

Norm	Euro 3	Euro 4	Euro 5 a	Euro 5 b	Euro 6
Type approval	since 2000	since 2005	since 2009	since 2011	proposed for 2014
Initial registration	since 2001	since 2006	since 2011	since 2013	proposed for 2015
CO	640	500	500	500	500
(HC + NOx)	560	300	230	230	170
NOx	500	250	180	180	80
PM	50	25	5	5	4,5
PN	–	–	–	6E11 /km	6E11 /km <sup>19</sup>



**Figure 18:** Modern after-treatment of exhaust gas systems<sup>20</sup>

### 2.2.3.1 Calculation of Weighting Tables for Integer Programs

In order to transform the operational profiles, one needs to calculate the revolution frequency  $\nu$  by evaluating the velocity  $v$  of the car and gear position  $g$ . So, the revolution frequency of the engine is given by

$$\nu = \frac{240 \cdot v \cdot g}{2\pi \cdot 0.3677} \cdot \frac{1000}{3600}. \quad (39)$$

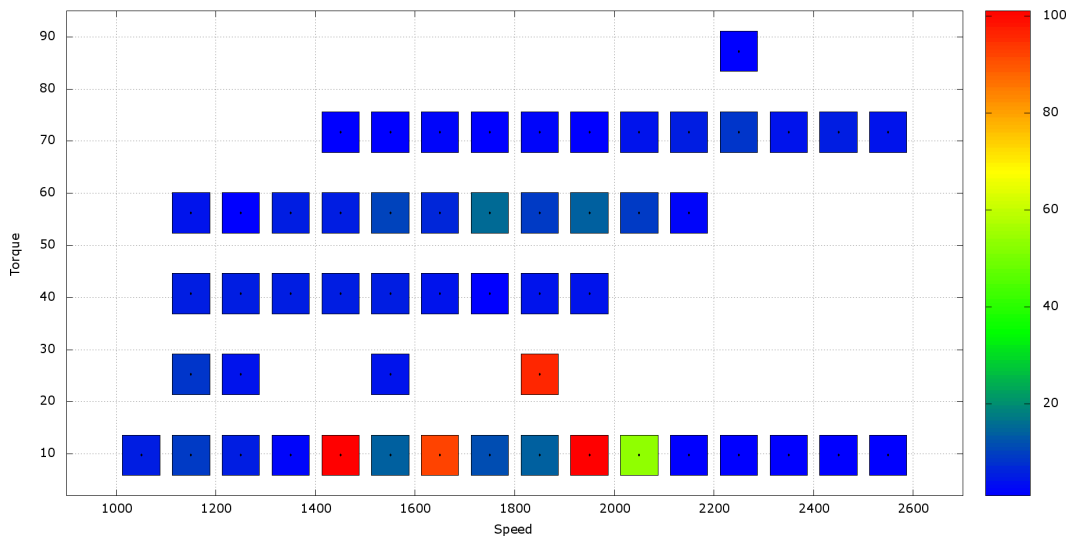
This empirical function (Equation (39)) results from a non-linear fit of a typical gear ratio profile.

Now, we have to determine the requested engine torque  $M$ . 10 Nm is the defined minimal torque, which is assumed to be necessary to drive the valves and crank drive. The average speed  $v_a$  between two time steps  $t_1$  and  $t_2$  is given by:  $v_a = \frac{1}{2} \cdot (\frac{v_0}{3.6} + \frac{v_1}{3.6})$  with the velocity  $v_1$  at  $t = t_1$  and velocity  $v_2$  at  $t = t_2$ . The requested engine torque can be approximated by the computation of the engine power  $P$ .  $P$  is given by

$$P = (0.625 \cdot v_a^2 \cdot cw \cdot A_{head} \cdot s_1 + cr \cdot m \cdot 9.81 \cdot s_1 + m \cdot \frac{1}{t_1 - t_0} \cdot \frac{v_1 - v_0}{3.6} \cdot s_1) \cdot \frac{1}{3600000 \cdot (t_1 - t_0)} \quad (40)$$

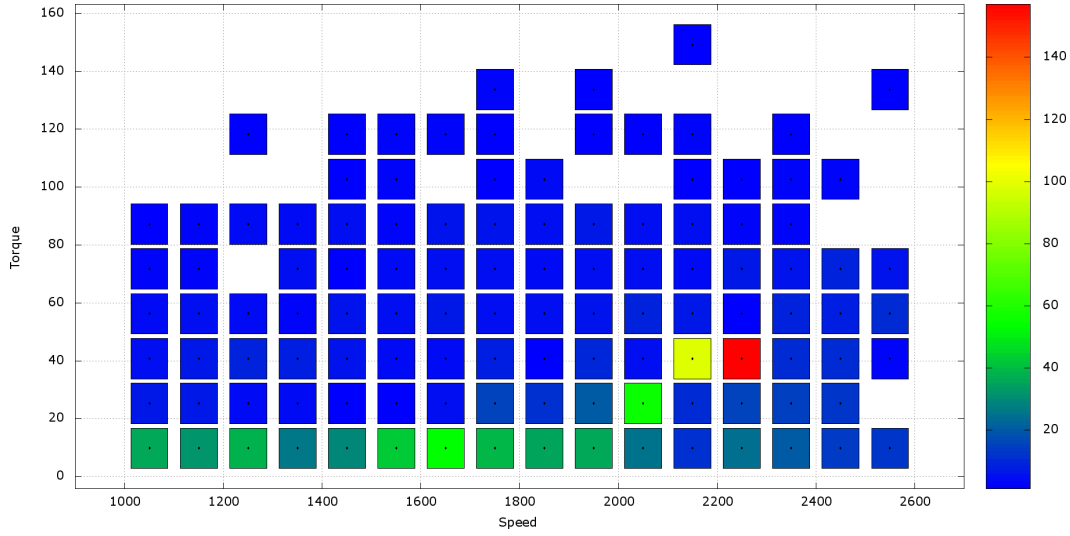
where  $cw$  is the flow resistance,  $A_{head}$  is the exposed head area,  $cr$  represents the roll drag,  $m$  is the mass of the car,  $s_1$  is the covered distance of last time step,  $v_i$  is the velocity at  $t = t_i$ . Finally, we are able to approximate the requested engine torque  $M$ :

$$M \approx \max(10, P \cdot \frac{9550}{\nu} \cdot 3600) \quad (41)$$



**Figure 19:** Weighting table of the NEDC discretized in 1 second time steps. Engine torque (Torque) in Nm, revolution frequency (Speed) in 1/min.

The resulting weighting tables of the NEDC and Random cycle are given in Figure 19 and 20. These Figures show the mean residence time MRT of the engine in operation points,



**Figure 20:** Weighting table of the RANDOM cycle discretized in 1 second time steps. Engine torque (Torque) in Nm, revolution frequency (Speed) in 1/min.

which are defined by revolution frequency and engine torque  $O_v^M = (v, M)$ . The integral limits for  $NO_x$ ,  $HC$  and  $CO$  in  $\frac{g}{cycle}$  is determined by the path length of the test procedure and the emission limit in  $\frac{mg}{km}$ . The evaluation of the emission integral value  $E_j$  of emission species  $j$  is given by the sum of all emission values of each operation point  $o \in O$  and its MRT in this operation point:

$$E_j = \sum_{o \in O} u_{ij} \cdot d_j^i \quad (42)$$

where  $d_j^i$  is the measured emission value of data point  $d_i$  of emission species  $j$  and  $u_{ij}$  is the weighting factors of data point  $d_i$  of quantification cycle of emission species  $j$ .

The Random cycle covers around 20 km, whereas the NEDC contains 11 km. Additionally, the NEDC is limited to 90 Nm, the Random cycle goes up to 160 Nm. In general, the Random cycle covers much more operation points. The NEDC prefers 5 operation points, in contrast to the Random cycle, which stresses around 15 points. Furthermore, the standard deviation of all MRTs is much smaller than in the NEDC.

#### 2.2.4 Stationary, Transient and Quasi Stationary Measurement

The measurement error of a signal  $\sigma_{total}$  is given by the statistical  $\sigma_{statistical}$  and the systematic error  $\sigma_{systematic}$

$$\sigma_{total} = \sigma_{statistical} + \sigma_{systematic}$$

The statistical error is caused by white noise of the sensor and other random based effects on the measurement signal. The systematic error is given by the experimental setup itself. Typical systematic errors are thermal disequilibrium, sensor offsets, malfunctions and unintended interactions with the environment. In our specific case, the most common systematic error are thermal drifts and the lagging of sensor signals.

Thermal drifts are given by the thermal system of the engine, since many parameters of the engine depend on temperatures, e.g. oil temperature. Furthermore, many monitored temperatures of the engine depend strongly on actuator settings, which are varied during

---

the measurement process. Lagging sensor signals or retarded signals can be caused by thermal effects, too. Moreover, signal lags are given by emission detection and finally by the physical latency of the engine, itself.

This work differentiates between the different measurement types. One way to classify measurement processes is to study the resulting total error of the measured signals. In a stationary measurement process a data point will be valid, if the statistical error is below a low limit, e.g.  $< 1\%$ . Additionally, the first derivative of the signal function is in average close to zero, so no systematical drifts are observed. The measurement time of a stationary value is typically up to 5 minutes.

A transient or highly dynamic measurement does not have the intention to derive highly accurate measurement values, but transient effects of the system. Therefore, actuator jumps are performed. The nearly instantaneous displacement of the engine system, leads to an oscillation process of the whole physical system. The quasi stationary measurement has been defined in this work. This kind of measurement is comparable with the stationary measurement, but limits the measurement time to a few seconds. Therefore, this method delivers more measurement points in the same time, but with a much bigger error bar than the stationary measurement. Waiting time between measurement points is longer than in highly dynamic measurement, but much shorter than in stationary measurements. The measurement time per measurement point depends on the requested maximal error, that is requested in order to solve the specific optimization problem.

---

## 2.3 Applied informatics

---

### 2.3.1 Binary Heaps

---

Binary heaps are implemented to manage priority queues efficiently. In this work, binary heaps are implemented in *Dynamic Testdrive*, in order to improve the performance of the Dijkstra routing algorithm (see Subsection 4.2.2).

The following definition and explanation of binary heaps are taken from [Kru11] pp 10 - 15. A binary heap is a data structure. In particular, a binary heap is a nearly complete binary tree with additional properties. A binary heap  $A$  is given by its length ( $length[A]$ ) and the number of stored items in the heap ( $size[A]$ ). Before, the first element can be stored in the binary head, we have to determine the maximal number of elements ( $length[A]$ ), that can be stored in the data structure. Every node  $i$  has a well defined relationship to its parent node  $parent[i]$ :

$$parent[i] := \lfloor \frac{i}{2} \rfloor \quad (43)$$

Furthermore, the descendants ( $left[i], right[i]$ ) of each node  $i$  are structured by:

$$left[i] := 2i \quad (44)$$

$$right[i] := 2i + 1 \quad (45)$$

The most important property of binary heaps is given by Equation (46).

$$A[i] \geq A[parent[i]] \forall i \in [1, size[A]] \quad (46)$$

So, the smallest element is always the root element of the binary tree. So, the heap is min-sorted. In the same way, it is possible to organize the binary heap max-sorted by reversing the greater-equal sign of Equation (46). In this case, the maximal element is the root element of the binary heap.

Figure 21 shows an example for binary heaps. The linear array on the left side is transformed into a binary heap on the right side, by applying the former introduced properties to the data set. The height of a binary heap  $A$  with  $size[A] = n$  is  $\lfloor \log_2 n \rfloor = \mathcal{O}(\log n)$ . Furthermore, the maximal number of elements on a specific height is given by  $2^h$ . Additionally, each height  $h'$  has to be filled completely, before a new level  $h$  can be created.



**Figure 21:** Illustration of binary heaps.<sup>21</sup>

---

<sup>21</sup> Source: [Kru11], p. 10, Figure 2.3

A new item is added in the end of the binary heap at index  $n + 1$ . So, the new item is descendant of node  $\lfloor (n + 1)/2 \rfloor$ . The new item has to be compared to it's parent and it has to be swapped with it's parent node until the parent node has a smaller value than the new item. This process is often called *Bubble Up*. Since the height of a binary heap of  $n$  elements is  $\mathcal{O}(\log(n))$  the effort of adding a new element is given by  $\mathcal{O}(\log(n))$ . After the extraction of the smallest element (root element) the last element  $y$  of the heap is set as new root element. Afterwards, element  $y$  has to *sink* down the tree, until all heap properties are true again. This process is analog to the *Bubble Up* process. So, the runtime of extraction is of logarithmic order.

### 2.3.2 Beneath-and-Beyond

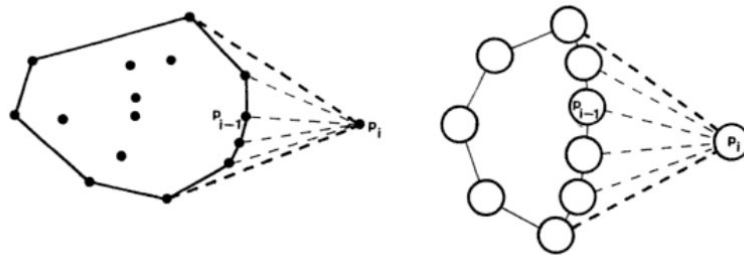
The *Beneath-and-Beyond* algorithm is a general dimension convex hull algorithm [Ede87]. Furthermore, the algorithm is iterative, so the convex hull is determined incrementally. This algorithm has been developed by Kallay et al. in 1981 [Mir11] and improved by several mathematicians, e.g. Grünbaum et al., Edelsbrunner et al.. The algorithm has been discussed by Edelsbrunner [Ede87] p. 143.

First, the data points have to be sorted lexicographically. So, the first iteration of the convex hull can be computed by the  $n$  first points in the sorted sequence.

$$\mathcal{P}_3 = \text{conv}(p_1, p_2, \dots, p_n) \quad (47)$$

Afterwards, the algorithm adds points  $p_i$  to the constructed convex hull  $\mathcal{P}_{i-1}$  point by point. A point is added by locating all visible facets of  $\mathcal{P}_{i-1}$  to the new point  $p_i$ . A facet is defined as visible to a point if the point is above the facet plane. This is illustrated in Figure 22.

The second step contains the determination of the horizon ridges of the new data point. A cone has to be constructed from the new point to the horizon ridges. The algorithm takes advantage of the idea, that a new data point is always outside the created convex hull, since the points are sorted lexicographically. So, one is able to define an open line segment that connects  $p_i$  and  $p_{i-1}$  but avoids the constructed convex hull. That is why the lines  $\text{aff}(p_i, v_t)$  and  $\text{aff}(p_i, v_b)$  are tangent to  $\mathcal{P}_{i-1}$  with  $v_t$  and  $v_b$  point in convex hull. in Figure 22 the line  $\text{aff}(p_i, v_t)$  is given by the connection of the new data point  $p_i$  and the upper end point of the computed polyhedron, the line  $\text{aff}(p_i, v_b)$  is given by the connection of the new data point  $p_i$  and the lower end point of the computed convex hull. So the new facets of  $\mathcal{P}_i$  have been created. Finally, all visible facets of  $p_i$  to  $\mathcal{P}_{i-1}$  have to be erased.



**Figure 22:** Illustration of principle of beneath-and-beyond algorithm.<sup>22</sup>

<sup>22</sup> Source: [Ede87], p. 144, Figure 8.3



---

In this work, we use an implementation of *Beneath-and-Beyond* in *Polymake* [Jos02] for validation and benchmark calculations.

---

## 3 Calibration Methods - State of the Art

---

### 3.1 Commercial Solutions

---

In the following section, we describe the state of the art of calibration methods for optimization of internal combustion engines (ICE). Many commercial products exist for measurement and calibration of ICE. The following section focuses on several subtopics and several commercial and research products.

---

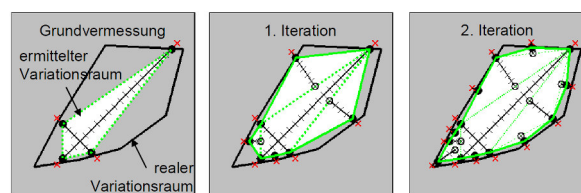
#### 3.1.1 Determination of data boundaries

---

In order to take measurements of the ICE, we need to know the exact boundaries of the operational space. Therefore, we determine hard and soft boundaries. Hard boundaries are for example cylinder pressures or critical device temperatures. Soft boundaries are non-destructive, but have to be conserved due to, e.g. emission and noise regulations. Thus, the data space is limited to several parameter settings, which does not violate boundary conditions. The detection of the operational space can be done by different techniques.

One classical method is the measurement via star like measurement paths with a defined step width. Each measurement path or ray will be interrupted if one observable violates a hard or soft limit. The convex hull of all measurement path points defines the operational space volume. This technique is typically automatized and part of the most commercial tools, like *Cameo - Online DoE Screening* and *TopExpert* [Gsc01].

The tool local linear model tree *LOLIMOT* contains a further method, developed by TUD and Porsche AG. This tool determines the boundary limits iteratively. The method is illustrated in Figure 23. The measurement starts with a coarse detection of the operational space. The measured boundary points define a convex hull. In the following iteration steps, the algorithm defines linear measurement paths, which are orthogonal to the facets of the convex hull. These measurement paths will be interrupted if a hard or soft boundary condition is violated [Ise00].



**Figure 23:** Illustration of iterative determination of operational space with LOLIMOT<sup>23</sup>

Besides the determination of the operational space, other programs monitor directly the boundary conditions. BMW developed the tool *mbminimize*, which uses regression models for engine limits [BMW03]. IAV created *Rapid Measurement*, which determines the stationary value, during the transient measurement by mathematical and physical models [Roe07].

---

#### 3.1.2 Test run planning

---

Stationary measurements are based on revolutions and torque of the engine, which define the operational points. The easiest way to cover an operational map is the complete grid measurement. Because of the increasing number of parameters this technique consumes a lot of

---

<sup>23</sup> Source: taken from [Ise03]

measurement time.

The *Full Factorial* method is the most obvious method, to study the systematic behavior of an unknown system. Since, this method is the most resource consuming, the benchmark is repeated several times, with increasing resolution. The total number of measurement points  $N_{max}$  is given by:

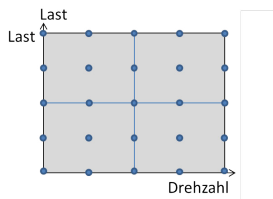
$$N_{max} = r^d \quad (48)$$

where  $r$  is the resolution of the measurement or the number of nodes per dimension  $d$ .

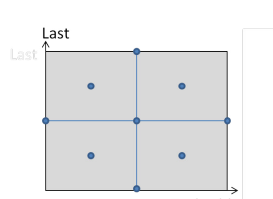
The most common technique for test run planning is the Design of Experiments *DoE* approach. The *DoE* is a stationary test run planning, which have to describe a mathematical or physical model with a minimal number of measurement points. These models are used for the final optimization. The classical test plans are summed up in CC-, space-filling and d-optimal designs (see Figure 24).

D-optimal test plans are optimal for the defined model, only. ICEs behave strongly non-linear, which lead to large deviations to mathematical models. An advanced version of D-optimal test plans are used by *LOLIMOT*. This tool creates local models via linear neuronal networks. This tool divides the operational space into subdivisions, which are represented by one neuronal network function [Ise10]. The tool *mbminimize* uses a d-optimal measurement plan, but uses a online-capability to decide which point to measure next. The criterion is given by the maximal information gain [BMW03].

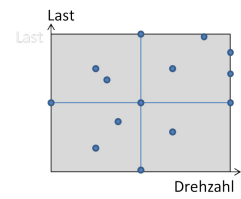
The Elring Klinger GmbH uses a totally different way for planning the tests. The test plan is given by a discretized norm cycle NEDC, i.e. the mean residence time of each operational point is computed if one would drive the test procedure. The temporal resolution is set to 1 second. Each operation point is weighted by it's relative measurement time. A relative distribution due to optimizing the emission is added to the existing measurement plan [Ste02]. This method is similar to the method applied in this work (see Subsection 2.2.3.1).



**Figure 24:** Full Factorial Design



**Figure 25:** CC Design



**Figure 26:** D-optimal Design

### 3.1.3 Measurement method

The measurement methods can be divided into stationary, quasi-stationary and transient/dynamic measurement. The stationary measurement method is most frequently used. Dynamic measurement techniques are still limited to very small application areas, e.g. the dynamic optimization of emission objectives [Ise00].

During a quasi-stationary measurement the parameters are varied slowly. Therefore the whole physical system follows the dynamic parameters with a defined contouring error. The

---

data acquisition records the whole system permanently. By varying one or more parameters, one observes the impact instantaneously. By inverting the measurement path it is possible to compensate the contouring error [Ise10]. A local model derivation is possible for up to 3 parameters [Ise10].

*DYNAMET* is a measurement strategy, in order to train dynamic neuronal networks. Therefore, the system is stimulated with a variation on every input signal. For non-linear processes one uses APRBS (Amplitudenmodulierte - Pseudo - Rausch - Binär - Signale) [Ise10].

---

### 3.1.4 Models

---

ICEs are difficult to describe by mathematical or physical models, which cover the whole operational space. Especially models for emission predictions are strongly limited. ICEs are so called MIMO (Multiple Input - Multiple Output)- Systems. Models often do not contain physical correlations, but describe the engine as black box system [Ise10]. If one would use models to predict observables, one has to know the prediction error. Afterwards, the optimization can be done on the model itself [BMW03]. Typically, one uses polynomial functions and *least squares* technique in order to approximate a simple model to measured data. These black-box methods works for known regims/engines only. Strongly non-linear regions are not describable by these approaches [Mit00].

A further approach are artificial neuronal networks (ANN), which work well for non-linear regimes. Furthermore, this black-box approach, does not need special knowledge about the ICE [Mit00]. Neuronal networks are separated mainly into Multi-Layer-Preceptrons (MLPs) and radial base function networks (RBF- networks). The principle of *LOLIMOT* is similar to RBF-networks [Ise10].

The tool *ASCMO* uses the superposition of weighted base functions. The weighting is derived by a statistical learning approach (SLA) automatically. This approach is robust and needs only a small number of data points to train the network [Kru09].

---

## 3.2 Preceding Stages of this project

---

### 3.2.1 Random Walker

---

The idea of adding the random walker principle (see 2.1.1) to *AmmOC* has been developed by M. Mertz et al. [Mer10] in 2009.

The symmetric lattice random walk principle had to be modified slightly. This walker algorithm has to derive optimization samples, so called incomplete local optima. Besides the orthogonal moving directions, the random walker is able to move diagonal.

Furthermore, the random walker moves over a two dimensional field *Sol\_Field*, given by revolution frequency and torque of the internal combustion engine. The structure of *Sol\_Field* is equal to the solution format of an electronic control unit (ECU). The data points of the  $n$ -dimensional data space, with  $n$  equal to the number of dynamic actuators of  $d_i \in D$  (see Equation (2)), are sorted into stacks depending on the location on *Sol\_Field*. Consequently, all data points within one stack share similar values for revolution frequency and torque (see Equation (4)), we call these stacks operation points  $O_v^M$  (see Equation (1)). In more detail, every data point component  $d_i^j$  of data point  $d_i$  in stack  $s_k$  share an equal value  $B_i^j$  (see Equation (49)<sup>24</sup> and Equation (50)<sup>25</sup>). The whole range of *Sol\_Field* is separated into  $I \in \mathbb{Z}^+$  equidistant intervals with bounds  $b_k^j \forall k \in [0, \dots, I-1]$  (see Equation (4)).

$$W_{bin} = \frac{b_0^j - b_{|b|}^j}{I} \quad (49)$$

$$B_i^j = \lfloor \frac{d_i^j}{W_{bin}} \rfloor \cdot W_{bin} \quad (50)$$

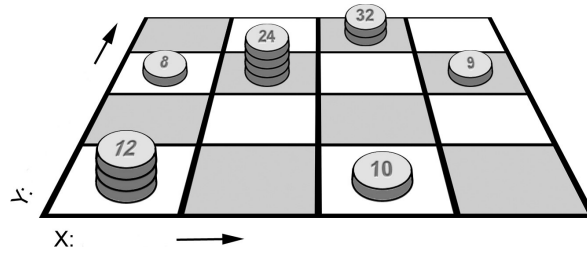
So, this algorithm adds data points to the sample, by choosing one data point of a stack. If the walker returns to a stack, which has already been visited during this run, the walker ignores this stack and keeps the already chosen data point. The random walker is not completely symmetric, it is not allowed to travel back to the point it has been in the previous step (direct back step). Arriving on an operation point  $O_v^M$ , the random walker algorithm selects one data point of stack  $s_k$ . In order to optimize the objective function, e.g. specific fuel consumption, the random walker takes the best data point in regard to the objective function, unless this decision does not violate any problem constraint.

An illustration of this method is shown in Figure 27. This figure shows a chessboard with stacks of coins. Each coin has its own specific value. All coins in a stack are sorted by its values, the highest value is up. The random walker enters a square and takes the coin with the highest value. While leaving the square, the random walker reminds this square as previous square. Afterwards, it enters the next square. Now, the algorithm tries to take the coin with the highest value of the new stack, again. However, the walker has to compare the new coin to the coin of the last square. Only if, all constraints are preserved, the algorithm is able to take this coin. Otherwise, it repeats the last step with the second coin of the stack, and so on.

---

<sup>24</sup> taken from [Mer10], Equation 3.9

<sup>25</sup> taken from [Mer10], Equation 3.10



**Figure 27:** Illustration of Chess board comparison. X and Y are given by definition of the solution field<sup>26</sup>

This decision process is strongly limited, the random walker compares the problem constraints with the previously taken data point, only. That is why, all solution constraints are preserved on the random walk path. Offside the derived path, it is possible, that constraints are not fulfilled, thus the transient solution is not completely feasible.

The problem constraints are given by the maximal solution gradient (see Equation (8)). These constraints are needed to compute transient solutions (definition given in Subsection 2.2.4). In other words, the algorithm derives solutions, that change actuators with a limited variation gradient, so that the physical system is able to follow the new actuator combination. The algorithm terminates if a minimal number of data points has been chosen. This threshold is defined by the user. Typically, this threshold is around 10 %.

Due to the fact, that the random walker algorithm derives incomplete local optima, the transient solution does not cover the whole solution width/area. Additionally, the constraints are only fulfilled on the random walk path. The big advantage of this algorithm is a very short run-time to derive a sample. Secondly, this process can run in parallel. A detailed description of this method and it's implementation can be looked up in [Mer10] chapter 3.41 and chapter 5.1.4. This algorithm has been developed, as preparation of the next algorithm "Deterministic Walker", which will be explained in the following subsection.

<sup>26</sup> Source: taken from [Mer10] Figure 5.24

---

### 3.2.2 Deterministic Walker

---

In consequence of the development of the random walker algorithm by M. Mertz et al. [Mer10], the algorithm of the deterministic walker has been evolved from it. The deterministic walker algorithm has been created by M. Gebhard et al. [Geb11] in 2011. The principle of the algorithm is based on support vector machines. The theory of support vector machines has been explained in Subsection 2.1.2.

The deterministic walker algorithm analyzes the incomplete local optima, which have been derived by the random walker algorithm. Afterwards, the algorithm combines and completes the optimum, but ensures, that all constraints are satisfied.

**Data Point Synthetization:** In contrast to the random walker algorithm, the deterministic walker algorithm needs a complete data space model or a data set with a high resolution. Therefore, two different approaches have been applied, the data rendering and the support vector model approach.

#### Data Rendering

The data rendering algorithm synthesizes data points on an user defined grid. All dynamic actuators (see Equation (2)) define the data space. The rendered data point component  $q_i^j$  is determined by all measured data points  $d_i \in D$  and a distance weighted average function  $w(\text{dist}(d_i, d_k))$ , where  $\text{dist}(d_i, d_k)$  is the euclidean distance between data point  $d_i$  and  $d_k$ :

$$q_i^j = \frac{1}{\sum_{k \in |D|, i \neq k} w(\text{dist}(d_i, d_k))} \sum_{k \in |D|, i \neq k} w(\text{dist}(d_i, d_k)) \cdot d_k^j \quad (51)$$

where  $d_i^j$  is component  $j$  of data point  $d_i$ .

A typical resolution is 20 supporting points per dynamic actuator, a typical number of dynamic actuators is 6 to 8. Thus the number of data points, which have to be synthesized is  $20^6 = 64,000,000$ . That is why, this algorithm has been realized in CUDA and OpenCL. This algorithm is able to synthesize data points in parallel.

#### Support Vector Model

The support vector model is an alternative approach in order to describe the data space [Geb11]. The theory has been explained in Subsection 2.1.2. The measured data points are used as training points for a support vector regression. The support vector regression is used as model, which delivers missing data points.

The distribution and measurement quality of the data points, cause support vector models with different systematical errors. In order to probe the systematical error, the support vector regression is trained on a randomly picked subset of the set of data points, only. Afterwards, the model can be tested by comparing the model values with data points that have not been used to train the model.

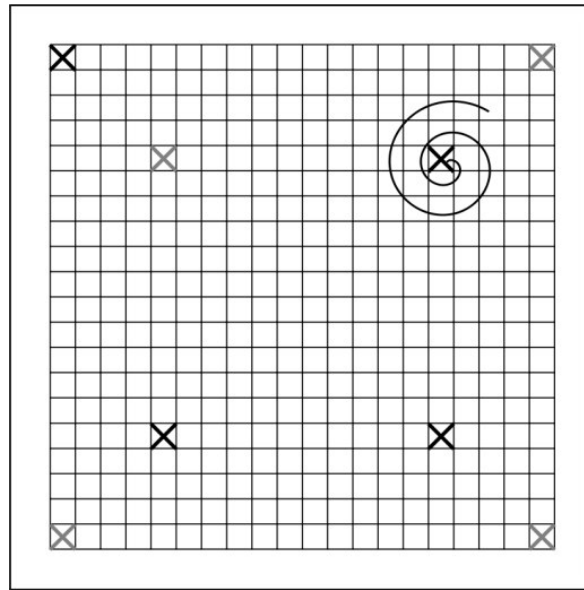
#### Modes of Operation of Deterministic Walker:

##### Engine map computation by seed points

The deterministic walker analyzes the local incomplete optima, which have been derived by the random walker algorithm. Therefore, the algorithm determines the data points, which have been chosen most frequent by the random walker for the local optima. So, it is possible to determine seed points by selecting a random subset of the most frequent data points of all local optima.

---

<sup>27</sup> Source: taken from [Geb11] Figure 4.4



**Figure 28:** Illustration of the construction of local optima with the Seed point method<sup>27</sup>

Figure 28 shows a typical start setup of a deterministic walker run. The configuration of seed points happens sequentially. Thus the algorithm checks, that the added seed point does not violate any constraint. After, all seed points have been added, the algorithm continues the solution on an analytic path. Every new point in the solution is checked against every existing data point in the solution. Therefore, the solution guarantees the integrity of the solution.

#### Engine map computation by random points

The random mode of the deterministic walker behaves similar to the seed mode. But, this mode does not add new points to the solution on base of an analytic path, but on randomly picked data points. Further information are given in [Geb11].



---

### 3.2.3 Greedy Ants

---

The *Greedy Ants* module has been designed and applied by P. Lind et al.. This subsection introduces this alternative approach for NP-hard optimization problems. For further details see [Lin12].

In principle, the algorithm can be compared to a family of ants, that are born in the starting point. Afterwards, each ant is looking for food with a limited number of steps. After a given time, all ants are compared and the most successful ants create more descendants, than the other ants. Finally, one ant family dominates and the marked path ways of this family is taken as optimization solution. Details are described in the following.

The *Greedy Ants*-algorithm is part of the family of evolutionary optimization algorithms. *Greedy Ants* are used to evaluate the optimization problem. Therefore it creates local samples of the global optimization problem. The problem description is given in Subsection 1.1. Feasible data points are defined by the critical constraint (see Equation (5)), the tolerance constraint (see Equation (8)) and by the maximal integral emission constraint in Equation (6). All data points  $d_i \in D$  have to be quantificated by comparison to other data points in the same region of the solution field, i.e. stacks (see definition in Equation (4)).

The evaluation of the tolerance constraint for each data point  $d_i \in D, i \in [1, \dots, n]$  is represented by the tolerance table  $T \in \mathbb{B}^{n \times n}$ . The nearest neighbors  $N \in \mathbb{B}^{n \times n}$  are determined by Voronoi tessellation (see Subsection 2.1.5). Before the optimization starts  $T$  and  $N$  are combined to the optimization graph  $P \in \mathbb{B}^{n \times n}$ . This is illustrated in Figure 29. The vertices of  $P$  are given by  $d_i \in D$ , the edges of this graph are defined by the possible transitions within the data points, according to the tolerance and nearest neighbor constraints. The tolerance constraints are illustrated in this figure by the maximal angle  $\alpha$ . The starting position  $i$  is given by a uniform random number out of the number of data points. At the first step this graph is generated by the following algorithm:

$$\begin{aligned} T_v &= T_{:,i} \mid \mid T_{i,:} \\ N_v &= N_{:,i} \mid \mid N_{i,:} \\ P &= N_v \ \&\& \ T_v \end{aligned} \tag{52}$$

where  $\vec{e}_i$  is the unit vector with component  $i$  equal 1,  $\mid \mid$  is the logical *or* operation on each component,  $\&\&$  is the logical *and* operation on each component,  $T_{:,i}$  represents column  $i$  of  $T$ ,  $T_{i,:}$  is row  $i$  of  $T$ .

The edges between the data points, show the valid transitions between valid data points, which can be chosen by the optimization program. A further extension of this concept is shown in Figure 30. So, the preprocessor combines the tolerance constrain, with the information of the nearest neighbors and the nearest neighbors of the nearest neighbors. This modification causes more feasible solutions, which can be found by the optimization algorithm.

If the optimization algorithm adds data point  $d_j$  to the solution after is has added data point  $d_i$  the available next data points are given by:

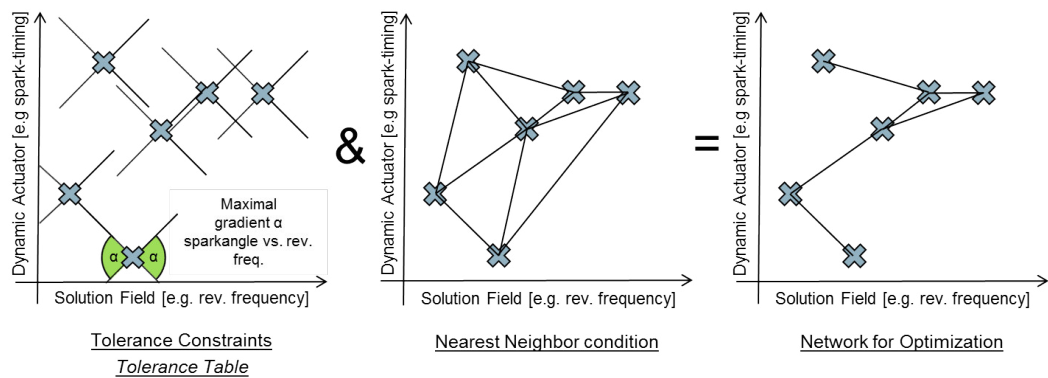
$$T_v' = T_{:,i} \mid \mid T_{i,:} \ \&\& \ T_{:,j} \mid \mid T_{j,:} \tag{53}$$

$$P = N_v \ \&\& \ T_v' \tag{54}$$

---

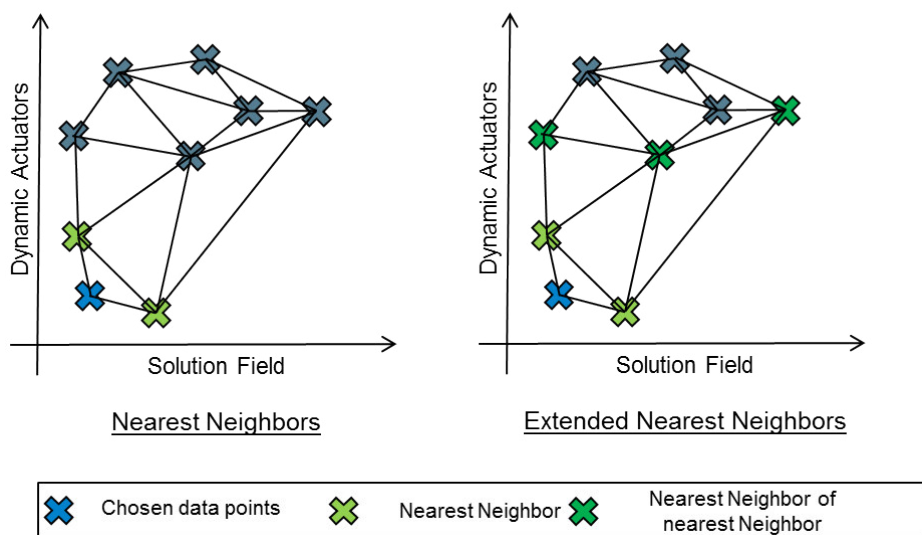
<sup>28</sup> Source: compare [Lin12] Figure 29

<sup>29</sup> Source: compare [Lin12] Figure 32



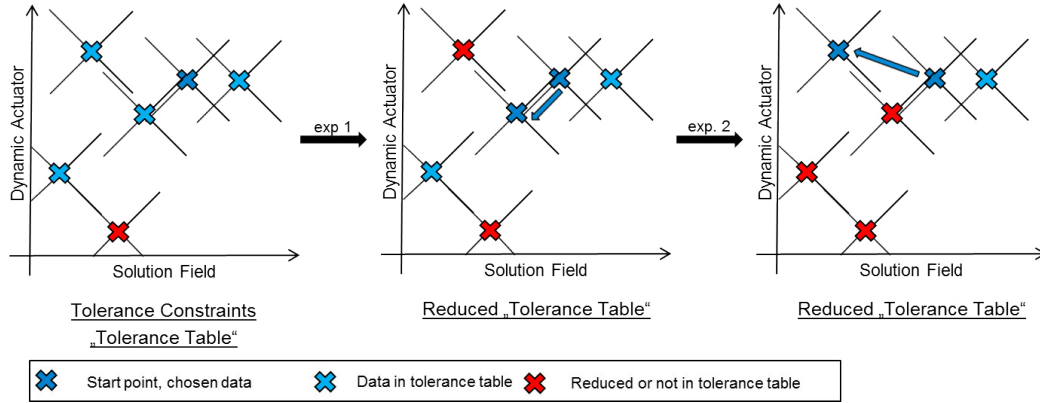
46

**Figure 29:** Components of the optimization graph for evolutionary program<sup>28</sup>



**Figure 30:** Illustration of nearest neighbors and extended nearest neighbors graph<sup>29</sup>

$T_v'$  is the reduced tolerance table, which contains the combined tolerance constraints of each data point that is part of the sample. Thus, the number of available data points decreases quickly with the number of chosen data points. That is why, the algorithm is able to derive samples quickly. The method of the reduced tolerance table is illustrated in Figure 31. This figure shows, that a chosen data point reduces the number of available data points. The reduction is caused by the re-evaluation of the tolerance constraints on the graph. So, the resulting solution fulfills the tolerance constraints, finally.



**Figure 31:** Illustration of reduction of tolerance table in graph<sup>30</sup>

Each time a data point is chosen, the corresponding entry in the objective function, which is given by the *prey* value in Equation (9), is stored for the final quantification of the ant's run. The selection process of new data points of the ant is repeated until the maximal number of steps is archived or no more valid data points for the ant are available (see Equation (54)).

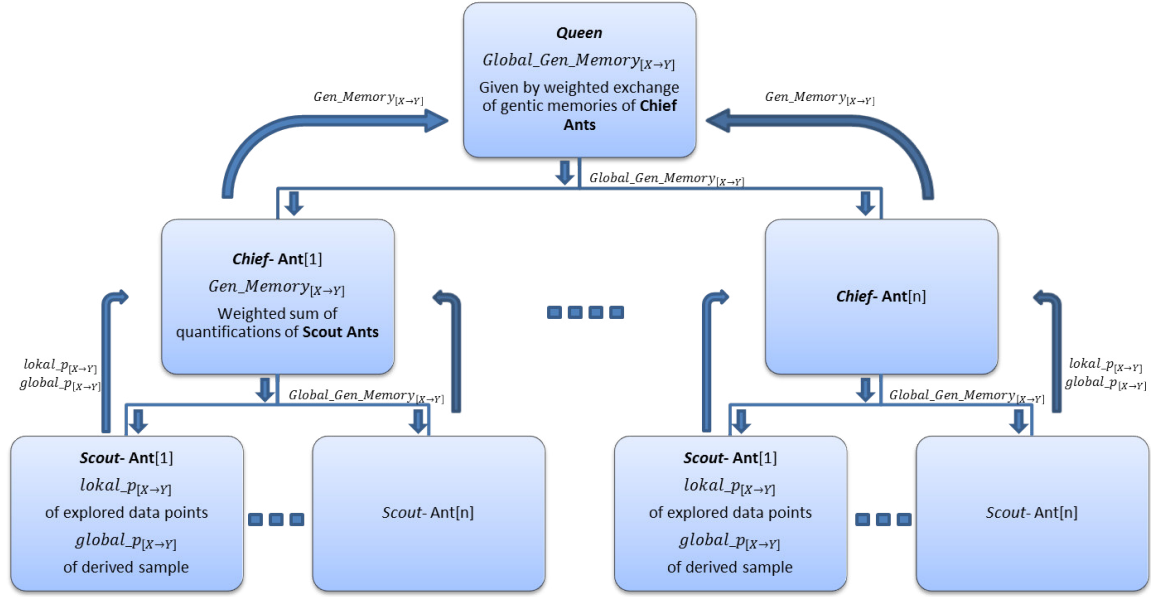
The genetic memory contains probabilities  $g_{ij} \in G \in \mathbb{R}^+ \times \mathbb{R}^+$  of transfer from point  $i$  to point  $j$  on the edge  $ij$ . The modification of the genetic memory is divided into two methods, the local and the global gratification. The local gratification is executed each time an ant enters a new data point. The genetic memory of this transfer is increased if the corresponding *prey* value is bigger than a defined parameter  $p_{limit}$ . So, the genetic memory is modified, in order to increase the possibility for making the same decision in the future. The genetic memory is modified by

$$g_{ij} += \text{int}((p_j - p_{limit}) / (1 - p_{limit})) \quad (55)$$

The algorithm includes a hierarchy structure, which keeps different parts of the genetic information. The hierarchy is shown in Figure 32. The *Ant Queen* is position independent. So, this part of the algorithm collects all genetic information, which have been collected by the *Chief Ants*. The collection of the queen includes a weighting function, depending on the quality of the found solution of each *Chief Ant*. So, genetic information is adapted in the global code in strong dependence of it's fitness. The chief layer is start position dependent. The start position is defined by random or by *Queen Ant*. The *Chief Ant* collects the local information of it's *Scout Ants*. These genetic information are combined with a global genetic imprinting by the *Chief Ant*. The criteria of this imprinting is equal to the criteria of the *Queen Ant*. The lowest layer, the *Scout Ants* mark local good data points, which provide a

<sup>30</sup> Source: compare [Lin12] Figure 30

high *prey* value. The local imprinting is visible to all ants in the network. This fact provides a system of cluster communication. Finally, the reproduction rate of *Scout Ants* is controlled by it's fitness value and the fitness value of it's *Chief Ant*. This defines the selection and reproduction mechanism.



**Figure 32:** Cluster communication: Setup of imprinting methods of the genetic memory<sup>31</sup>

The global gratification is executed after all ants terminated, so after all ants have no other data point to choose or the maximal number of steps have been achieved. The global method modifies the genetic memory of the *primary ant*. The *primary ant* is the genetic prototype of the current generation of ants, which take samples of the optimization problem. Each ant has been created from the genetic profile of the *primary ant*. In the global gratification method the achievements of each ant is re-directed to it's *primary ant* in order to keep the gained information for upcoming generations. The method judges each ant's fitness by the collected *prey* values  $P_i$  of ant  $i$ :

$$F_i = \begin{cases} 2 \cdot \frac{P_i}{\sum_{j=1, i \neq j}^n (P_j)} & \text{if } N_{found}^{norm} \geq N_{limit} \\ \frac{P_i}{\sum_{j=1, i \neq j}^n (P_j)} & \text{else} \end{cases} \quad (56)$$

where  $N_{found}$  is the number of collected *prey* values and  $N_{found}^{norm}$  is given by:

$$N_{found}^{norm} = \frac{N_{found}^i - \min(N_{found})}{\max(N_{found}) - \min(N_{found})} \quad (57)$$

<sup>31</sup> Source: compare [Lin12] Figure 34

Finally, this fitness value  $F_i$  is normalized and compared to the fitness of other ants, the fitness of all ants is given in the set of fitness  $F_i \in F \in \mathbb{R}$ :

$$F_i^{norm} = \frac{F_i - \min(F)}{\max(F) - \min(F)} \quad (58)$$

where  $\min(F)/\max(F)$  is the minimal/maximal fitness value in  $F$ .

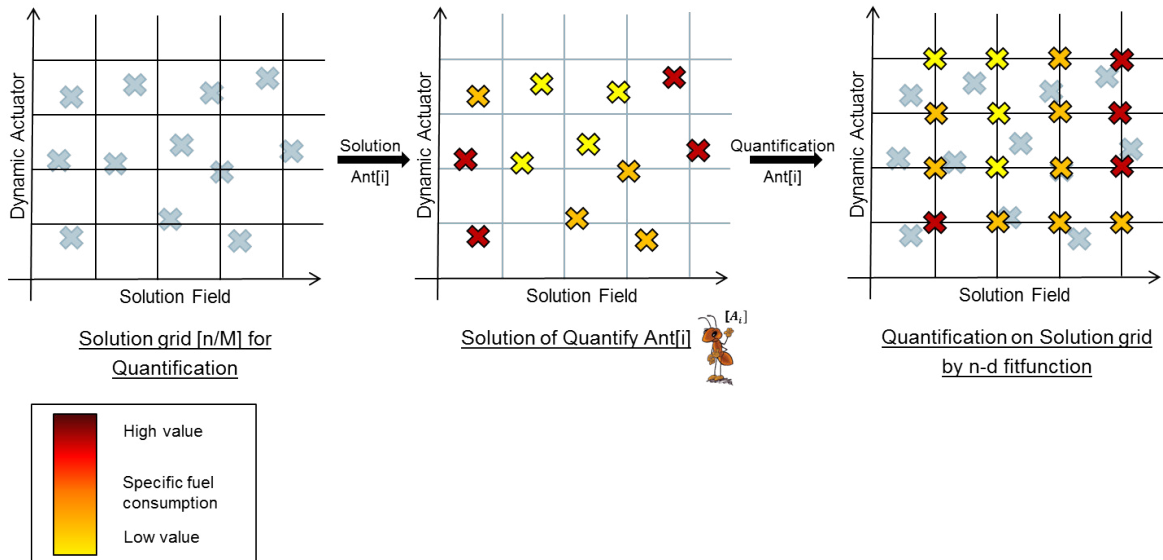
Ant  $i$  of  $N_A$  ants transfers its genetic imprinting  $G_i$  to the prototype ant of the next generation *primary ant* in accordance with its fitness value and with respect to the number of generation  $g$ . The new genetic imprinting of the prototype  $G_{primary}$  given by:

$$G_{primary} += (\text{int}(\frac{g}{10} \cdot F_i^{norm}) + 1) \cdot G_i \forall i \in [1, \dots, N_A]$$

Afterwards the next generation continues the optimization.

The quantification of samples helps to decide which sample is the best. The quantification is applied for all optimization algorithms, in order to maintain comparability. The produced samples of the *Greedy Ant* algorithm is heterogeneously distributed on the solution field. Classically, the solution field is given by the revolution frequency of the crankshaft and the requested torque of the engine (see Equation (1)). In order to improve comparability of samples the quantification is done on an equidistant grid on the solution field. The structure of the quantification field is given by the definition of the final solution resolution.

The transformation of heterogeneous samples into equidistant samples is done by two processes, the determination of the actuator settings and the fitting of the sensor values. The determination of the actuator settings is done by a distance weighted average function. This routine is equal to the refinement process of the *measurement specialization* (see Subsection 4.2.5). Thus, all actuator settings of the grid points are fully determined.



**Figure 33:** Quantification of found solutions by *Greedy Ants* algorithm<sup>32</sup>

<sup>32</sup> Source: compare [Lin12] Figure 37

---

The evaluation of sensor values on the grid points is done by a 2nd order polynomial fit at the determined actuator settings. The minimal number of data points is given by  $10 \cdot (d + 1)^2$ . The selection of data points is done by selecting the nearest neighbors of the requested grid point. If the number of neighbors is smaller than the minimal number, the neighbors of neighbors are selected. This is repeated until the minimal number of data points is achieved. The error of the fit has to be considered. The fitted result value is shifted by the half of sigma to worse result values, i.e. fuel consumption is moved up. Rarely covered samples provide higher fitting errors, so the quantification is worse than by a sample that covers the solution field better. The final quantification of a sample  $i$  is condensed into a single quantification factor  $q_i$ . This factor is evaluated by a combination of points of the quantification field. Each point on the quantification field does not have to contribute equally, since a quantification regards driver profiles or specific driving cycles (see Subsection 2.2.3).

---

## 4 Framework for the quasi stationary data acquisition and integer optimization

---

This section introduces and explains all methods that have been developed in this work. These methods handle the quasi stationary data acquisition (see Subsection 2.2.4) and the integer optimization of the measured data points. The whole project "*Automatic multidimensional, multicriterial optimization algorithm for the calibration of internal combustion engines*" has been abbreviated to *AmmOC*.

---

### 4.1 General

---

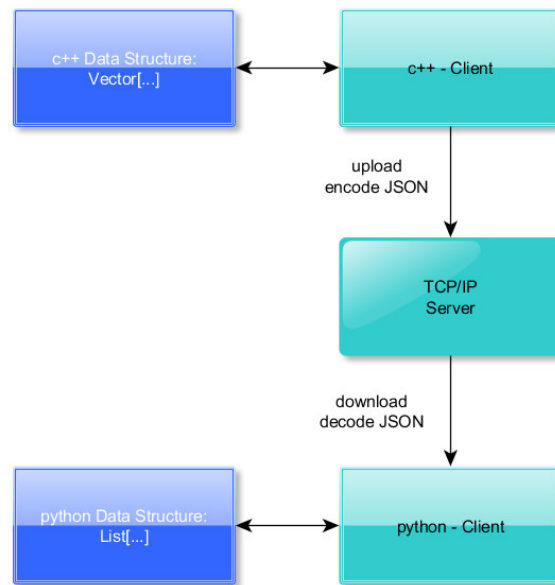
---

#### 4.1.1 TCP/IP Server

---

The *AmmOC* project contains source code written in perl, python, c, c++, openCL and CUDA. That is why *AmmOC* contains an own TCP/IP server. The server has been implemented in perl, its clients are implemented in c/c++, perl and python. So, it is possible to realize the inter-process communication (IPC) between the different parts of *AmmOC*. Each routine in the project is able to upload data structure, e.g. scalars, arrays, dictionaries and custom structure. The structures are serialized by JavaScript Object Notation (JSON)<sup>33</sup>. After the serialization the string is uploaded to the server hash structure.

This method is generalized and can be implemented in c/c++, perl and python in the same manner. Furthermore, data structures, which have been uploaded, can be used by any other routine. A small example is given in Figure 34.



**Figure 34:** Server Client Setup (excerpt) of *AmmOC*

In this example, a routine, written in c++, e.g. Greedy Ants (3.2.3) allocates a dynamic data structure, e.g. vector, which contains the data of the other process. Finally, this structure is uploaded and serialized in the c++ client module. Afterwards, the stored serialized structure is downloaded by the python client module and decoded to a python data structure.

<sup>33</sup> Source: <http://docs.python.org/2/library/json.html>

---

## 4.2 Data Acquisition and Cleaning

---

*AmmOC* contains data acquisition and cleaning functions. This work generates measurement plans, which define the measurement process at the engine test bench. During the measurement process *AmmOC* does plausibility checks, furthermore it detects semi-automatically controller-oscillations within the measured time series. The critical period is automatically marked, the user is able to erase the marked periods. In the same time, *AmmOC* compresses the measured data, in order to accelerate further analysis steps.

The data processing, plausibility and oscillation checks are done by *b.Osco*, which has been developed by M. Mertz et al. [Mer10]. *b.Osco* processes the raw data. It checks general parameter limits and oscillations with a linear regression approach (see more in Subsection 2.1.6). *AmmOC* provides several ways for the data room description, because of different types of optimization methods. It is possible to optimize on the basis of continuous functions or on the basis of discrete data points. Therefore, one needs different data space descriptions. *AmmOC* uses hyper plane rendering or support vector regression algorithms for the continuous description of the data space. Within the hyper plane rendering algorithm, *AmmOC* calculates any position of the data space by distances-weighted average values in reference to all measured data points. Thus, the interpolation method depends on the distances-weighting function of the average-value-function. The support vector model (SVM) approach creates a data model on the basis of all data points. This neural net algorithm trains itself on the measured data. Afterwards the neural net is able to interpolate values for a continuous description. The SVM is mainly defined by the number of iteration steps and the initial gaussian width. For the discrete description, *AmmOC* uses the explicit method of discretization via Voronoi tessellation. Each measured data point is seed point of a Voronoi region. Voronoi regions are defined as sites with the following properties. Each point within the Voronoi region is closer to the seed point (VSP) of this cell, than to any other VSP. Thus, one receives an explicit discretization of the whole data space, without any overlapping. Furthermore, one gains information about the nearest neighbors of a measured data point in the highly dimensional space. Consequently, one is able to derive the local gradient of parameters with respect to its nearest neighbors. These gradients are used by the online-capability. Thus, it is possible to refine measurement maps in regions of large gradients in the observables. Additionally, the homogeneity of the volumina of each Voronoi cell is connected to the homogenous coverage of the operational space by the measurement process. *AmmOC* creates local predictive models within the Voronoi cells, in order to compare newly measured data to recent measured data. *AmmOC* focuses on the derivation of newly measured data to the predication by older values, in order to refine the measurement in regions with large deviations and in order to coarse the measurement in regions with rather simple correlations. The derived prediction error can be used as cancel criterion of the measurement process. Before the measurement takes place, a partial measurement plan has to be created. The measurement plan contains typically one to 10 measurement ramps. After finishing a partial measurement plan, the data processing continues. If the basic calibration achieved the wanted quality, the program terminates, else a new partial measurement plan will be created. Each of these introduced topics will be discussed in detail in the following paragraphs.



---

#### 4.2.1 Data Space and Solution Field Organization

---

The raw data stream of the measurement is organized in two data trees, the data tree, which is structured by all dynamic actuators *Tree* and the data tree, which organizes the solution field *Sol\_Tree*. In principle, the data tree is an incomplete tree, which gains  $2^d$  leaves on each splitting process, where  $d$  is the number of dimensions of the tree. The splitting methods are described in a following subsection.

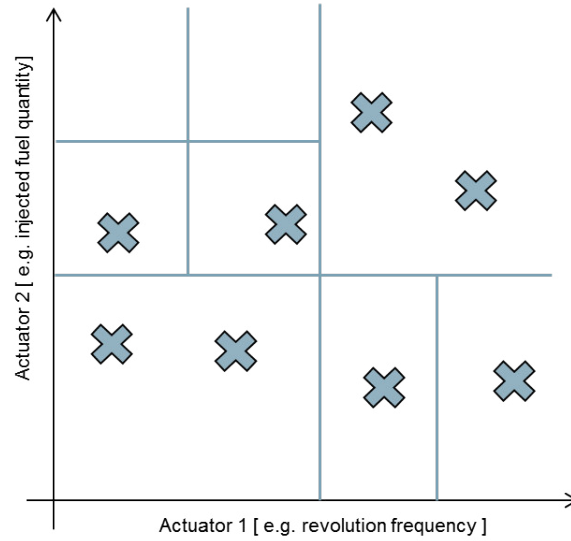
**Table 4:** Bins of an example tree for different tree depths

Tree depth	Bins
1	[650, 2600]
2	[650, 1625] [1625, 2600]
3	[650, 1137.5] [1137.5, 1625] [1625, 2112.5] [2112.5, 2600]

Table 4 shows an easy example for a symmetrically split data tree. Each bin  $h_k \in H$  is given by its center coordinates, which is given by an array with  $d$  absolute actuator values, where  $d$  is the number of dimensions of the data tree, furthermore, each bin contains additional information, e.g. number of events and best value. These additional values are introduced in the corresponding subsections. In this example, we handle an one-dimensional data tree with a minimal value 650 and a maximal value 2600. The first level of the tree is given by one data bin, which covers the whole range. At the second level, the whole range is separated into two intervals, 650 to 1625 and 1625 to 2600. The last layer contains four bins, which originated in the two bins of level 2. In order to adapt to more complex structures it is necessary to produce asymmetric data trees, an example for these more complex trees is given in Figure 35. This data tree has two dimensions, i.e. revolution frequency and injected fuel quantity. The first level split divided the whole data space into four quadrants. Secondly, the upper left quadrant is split again. The first two splits are homogeneous splits. The last split in Figure 35 is an asymmetric one of the lower right quadrant. In this case only a subset of actuators are refined by the splitting process.

By splitting the data space tree, *AmmOC* adopts to the structure of the optimization problem. Regions of low interest contain larger bins, relevant sections of the data space are divided into finer bins. The splitting mechanism will be explained in detail in Subsection 4.2.5.

In this work, we create a data tree *Tree*, which is given by all dynamic actuators. In this data tree, we structure all data points of the measurement, furthermore, the routing algorithm navigates within the bins of *Tree*. During the measurement process, we create a second data tree *Sol\_Tree*. This tree is given by the dynamic actuators and result values, which are part of the solution field as defined in Subsection 1.1. Typically, the solution field is given by the revolution frequency and the generated torque of the engine. In each bin of *Sol\_Tree* we store the highest value of the objective function (see Equation (9)). We call *Sol\_Tree* the unconstrained solution of the mathematical problem of Subsection 1.1, since we neglect the emission constraints. Furthermore, we store the best values of each sensor, which contributes to the objective function or to the integral emission side constraints (see



**Figure 35:** Illustration of data tree concept. Blue grey lines represent borders of the data tree.

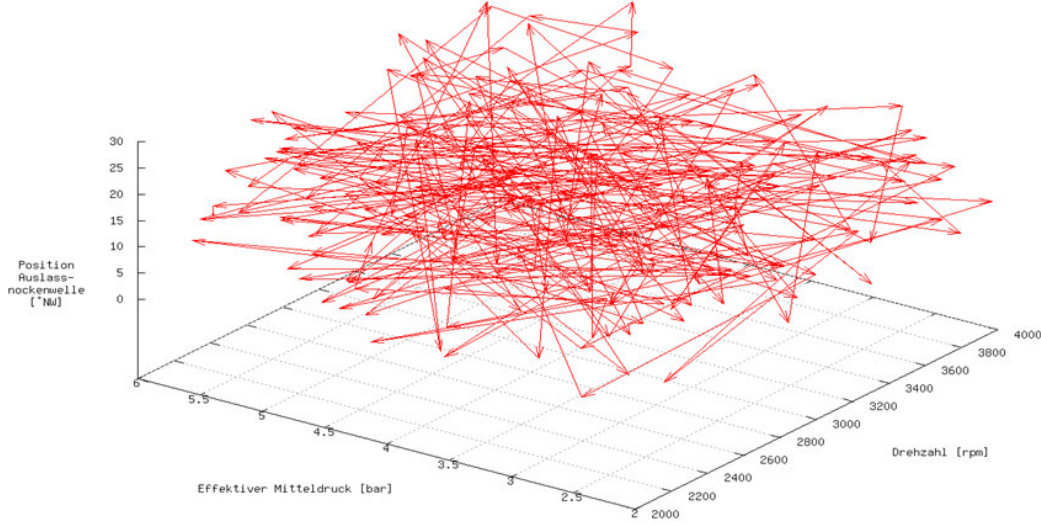
Equation (6)). Typically, this includes besides the specific fuel consumption, the emission rates of all exhaust species.

#### 4.2.2 Dynamic Test Drive

The creation of measurement plans in *AmmOC* is done by *Dynamic Testdrive*. The created plans consist typically of more than 6 parameters and cover homogeneously the whole operational space in a defined amount. Furthermore, it is possible to adapt the measurement density to the local gradients, during the measurement process. The homogeneous coverage of the operational space is realized by weighted random numbers. The randomly drawn parameter settings define the start- and end- positions of a measurement ramp. Measurement ramps are given by linear functions for all defined dynamic actuators with a defined slope of each actuator. Since, the operational space consists of more than 6 dimensions, the weighting function consists of the same amount of dimensions. The weighting functions are modified by the analysis of the data points, thus regions with high gradients or high objective values are preferred. On the basis of the new weighting function, one draws randomly new start- and end- positions of new measurement ramps.

This work uses a quasi-stationary measurement approach. Quasi- stationary means, that the holding time of the measurement is much shorter than in the stationary measurement process, but the measured value reaches a well-defined relative fraction of the stationary value, i.e. 95%. Because of the smaller parameter gradients than in the transient mode, it is possible to conserve the operational limits of the engine, without extrapolation or prediction. In contrast to the transient measurement mode, the quasi-stationary approach does not use step function responses. During the measurement, the algorithm is able to analyze the measured data and to define a criterion to stop the measurement process in a specific region of the operational space. Thus, measurement time can be reduced. The cancel criterion is given by a convergence value, which is defined by the deviation of the quasi-stationary and the stationary value, which is derived in a defined measurement pre-evaluation. For this purpose, *AmmOC* defines measurement plans with a variety of maximal gradient settings in specific regions of the operational space. These methods will be explained in the next section.

#### 4.2.2.1 Density function of data points



**Figure 36:** Illustration of linear measurement ramps.

The function *Dynamic Testdrive* handles the creation of partial measurement plans. The new partial measurement plan continues from the last data point of the previous measurement plan. The creation of measurement plans is done by the analysis of the corresponding data tree (see Subsection 4.2.1). Each bin contains all previously measured data points  $d_i \in D$ , which are within the borders of the bin  $h_k \in H$ , where  $H$  is the set of all bins in the corresponding data tree. In order to create a density function of the measured data points, we analyze the number of events in the data tree *Tree*.  $n_k$  is the number of data points in bin  $h_k$ . In order to determine the weighting function for the generation of random numbers, we have to invert the data tree histogram  $H$  for each bin. Therefore, we determine the average frequency  $\bar{n}$  of all bins  $h_k \in H$ . The inverse value of each bin  $n_k^{inv}$  is given by:

$$n_k^{inv} = 2 \cdot (\bar{n} - n_k) + n_k \quad (59)$$

Afterwards we define the inverted tree histogram  $H^{inv}$  by the center coordinates and borders of each bin  $h_k \in H$  but with the inverted number of events of Equation (59). Each bin in the inverted tree histogram  $H^{inv}$  is reduced by the smallest frequency of all bins in  $H^{inv}$ . Consequently, we obtain at least one bin with its value equal 0, other bins contain values greater 0.

The next work step is the drawing of random numbers according to the inverted tree histogram. Therefore every *bin* of the tree histogram is written into an array *he*. Each bin in *he* is given by its center coordinates. In a second array *hf* we write the entry of the corresponding bin of the inverted tree histogram. The algorithm draws uniformly distributed random numbers with variable seeds between 0 and the total sum of all entries in the inverted tree histogram. This random number  $R$  has to be reduced by the first value of *hf*. If  $R$  is still greater than zero the reduction continues with the next value of *hf* and so on, until  $R$  is smaller than 0. Finally *he* of the last index of *hf* which has been subtracted of

$R$  is the selected bin of  $he$ .

Example for a two dimensional tree histogram:

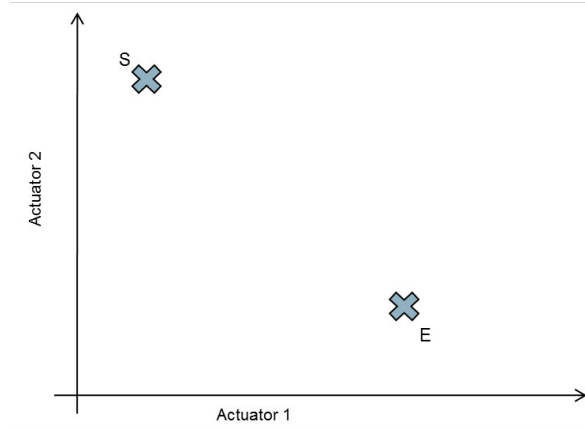
$$\begin{aligned}
 he &= [[1, 1], [1, 2], [2, 1], [3, 2], [2, 3]] \\
 hf &= [0, 5, 7, 1, 2] \\
 l_{lower} &= 0 \\
 l_{upper} &= \sum_{i=1}^5 (hf_i) = 15 \\
 R &= 12 \\
 he(selected) &= he(4) = [3, 2]
 \end{aligned} \tag{60}$$

In this example the next measurement ramp would go from the current position to position  $[3, 2]$ . This means actuator 1 will be set to 3, the second actuator will go to 2. The described sequence is illustrated in Figure 37 and Figure 38.

The measurement ramp is a set of discrete actuator combinations. *AmmOC* distinguish between static and dynamic actuators. Static actuators have a fixed value, which will not change during the whole measurement, dynamic actuators change during measurement. Therefore, a measurement ramp consists of fixed and variable actuator settings. A measurement plan has a configurable measurement frequency  $\nu_{meas}$  and a configurable recording time  $t_{meas}$ . Furthermore, each dynamic actuator  $j$  has a maximal test bench gradient  $\overline{\nabla R}_{tb}^j$ . A measurement ramp is a sequence of actuator combinations that depend on the previous combination. Let  $i$  be the index of the current combination in the sequence, so the new component  $j$  in combination  $i$  is defined as:

$$\begin{aligned}
 d_i^j &= d_{i-1}^j + \overline{\nabla R}_{tb}^j \cdot (1/\nu_{meas}) \\
 t_i &= t_{i-1} + (1/\nu_{meas})
 \end{aligned} \tag{61}$$

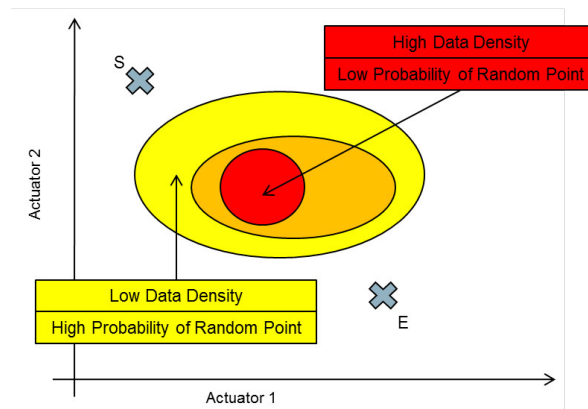
The creation of new actuator combinations stops if the measurement time  $t_i > t_{meas}$ .



**Figure 37:** Derive measurement ramps: uniform random numbers

#### 4.2.3 Measurement routing

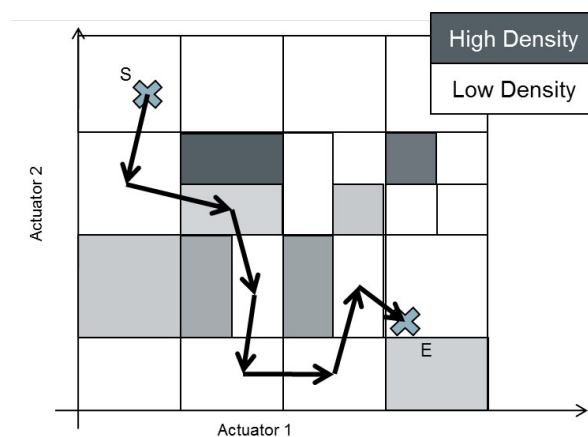
The implementation of the *Dijkstra* algorithm has been taken from [Epp02a]. The implementation of binary heaps have been taken from [Epp02b]. The measurement routing is an



**Figure 38:** Derive measurement ramps: Weighted random numbers by inverse measurement density

additional feature of the *Dynamic Testdrive* module. Whereas the previous functions derive start- and end-points, this function handles local inhomogeneity. Local inhomogeneities are given by regions with a high data density. In order to bypass these regions, this algorithm interprets a high data density with additional costs, which are minimized by the Dijkstra algorithm.

The method is illustrated in Figure 39. The derived start- and end-points are connected with a piece-wise linear path. Corresponding to the shortest path method, the algorithm bypasses data space regions with a high data point density. The data space partitioning is done, as in the previous subsection, by the data tree *Tree*. In Figure 39, we see, that the piece-wise linear path passes each center of the selected bin  $h_k$  of  $H$ . In order to obtain a better homogeneity within a bin  $h_k$  of  $H$  the routing algorithm applies uniform random numbers to vary the destination point within the *Tree* site. The handling of local inhomogeneities improves the global homogeneity. Furthermore, the selection of equal end-points cause different paths.



**Figure 39:** Derive measurement ramps: Get piece wise linear path by Dijkstra Algorithm

---

## 4.2.4 Data Cleaning

---

The data cleaning algorithm is supposed to erase data points, that violate upper or low limits, e.g. critical temperatures, sensor thresholds, etc.. Afterwards, the signal quality has to be improved and the total number of data points has to be reduced, since the evaluation time of our optimization algorithms depend strongly on the number of data points.

The raw data points are derived from two different reference problems, which are presented in section 7.2. Both models compute stationary sensor values for a given combination of actuator and parameter settings. These sensor values do not contain statistical or systematic errors. For this reason, the data points  $d_i \in D$  have to be equipped with an artificial error  $x_i$ , so that  $d_i^{real} = d_i + x_i$ . In consequence, we are able to test noise reduction methods. The error of the observables of interest  $x_i$  is given by a statistical error  $x_i^{noised}$  (Equation (62)), i.e. Gaussian white noise and a systematical error  $x_i^{retarded}$  (Equation (63)), i.e. signal retardation. The statistical error is caused by an oscillating and a random contribution. The amplitude of the oscillation  $A_\omega$  has a circle frequency  $\omega$ . The random or *white noise* contribution is generated by a function  $x_{Random}(-A_{white}, A_{white})$  that returns uniform random numbers between  $-A_{white}$  and  $A_{white}$ ,  $t_i$  has been defined in Equation (61).

$$x_i^{noised} = d_i \cdot (1 + A_\omega * \sin(\omega t_i)) + x_{Random}(-A_{white}, A_{white}) \quad (62)$$

The systematic error  $x_i^{retarded}$  depends on the previous data point  $d_{i-1}$  and a maximal relative gradient of this actuator  $f_{lag}$ :

$$x_i^{retarded} = d_{i-1} + (d_i - d_{i-1}) \cdot f_{lag} \quad (63)$$

Finally, the complete error of data point  $d_i$  is given by

$$x_i = x_i^{noised} + x_i^{retarded}.$$

### 4.2.4.1 Hard boundary filter

The hard boundary filter is a simple upper and lower threshold cutter. The user configures maximal and minimal valid values for actuators and observables. If a boundary condition of any threshold is violated, the data point is marked as invalid. The hard boundary filter satisfy the constraint introduced in Equation (5). A data point  $d_i \in D$  does not have to be erased if each component  $d_i^j, j \in [1, .., |d_i|]$  is within it's defined lower  $\underline{p}_j$  and upper bounds  $\overline{p}_j$ :

$$d_i^j \in [\underline{p}_j, \overline{p}_j] \forall j \quad (64)$$

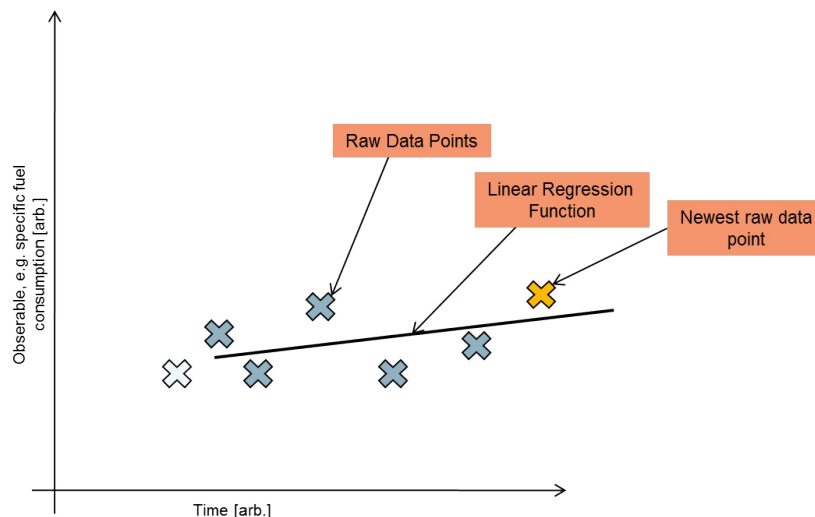
This filter prevents severe damage from the combustion engine. Typical critical observables are turbocharger, catalyst temperatures or inner cylinder pressure, but also maximal sensor values, which take part in the objective function. These observables are indicators for the driveable data space. Furthermore, this filter is able to focus the measurement on the reasonable region. The reasonable region is a subset of the driveable data space. with respect to the optimization criteria, some regions can be excluded directly, e.g. the emission levels are much to high. In these cases, the contribution of these data points is highly unlikely, since the maximal contribution of a single data point should not extend an user defined fraction of the global emission threshold. Therefore, it is reasonable to interrupt measurement ramps, that overcome defined emission or fuel consumption limits. If *AmmOC* is implemented on the real engine test bench and not on an engine model, the hard boundary filter would interrupt a measurement ramp, in case of boundary violation.

#### 4.2.4.2 Controller oscillation filter

Generally, an engine test bench is equipped with direct actuating variables, e.g. ignition angle, start of injection, and controlled actuating variables, e.g. revolution frequency, engine torque. If an actuating variable is controlled by any kind of controller, it is possible that the controller starts to oscillate. The oscillation is detected by monitoring the specific fuel consumption, since this is a very sensitive observable. The tool *b.osco* detects oscillations in the specific fuel consumption data stream and marks data points, if a subset of data points cannot be described with a moving linear regression. A subset of data points cannot be described with a linear regression, if the average root mean square deviation overcomes a user defined threshold.

#### 4.2.4.3 Determination of valid data points

The raw data points of the test bench or of the data model have to be processed in order to improve the quality and to reduce the total number of data points for further calculations. After the data points pass the boundary and the oscillation filter, the algorithm continues with a moving linear regression on a subset of the measured data points. The number of participants is constant and configured by the user. The *moving linear regression* function has been defined in Subsection 2.1.6.



**Figure 40:** Illustration of moving linear regression.

The applied principle of the *moving linear regression* is illustrated in Figure 40. The raw data points (blue) are described by the *moving linear regression* (black line). The white raw data point is the latest data point, which does not contribute to the *moving linear regression*, the yellow one is the latest data point, that contribute to the mechanism.

After the coefficients of the linear regression have been derived, we interpolate one data point in the middle of the selected time interval, this data point is called *valid* data point.

#### 4.2.4.4 Retarded Signal Handling

Observables of a combustion engine can be classified by different response times or the signal latency. One possible source for the signal latency is given by physical processes that modify the boundary conditions of the internal combustion process. These modifications do



---

not happen instantly. By varying an actuator, e.g. air throttle, the engine performs a number of burn cycles with non-constant boundary conditions. In the special case of the air throttle, the amount of air, which is available for the combustion depends not only on the throttle position, but also on the pressure gradient, caused by the combustion and the back-pressure of the exhaust system. Another different kind of retarded signals are temperatures in the exhaust region of the engine. Due to heat capacity of objects, an object, e.g. catalyst or turbocharger, does not adopt instantaneously to the current exhaust air temperature, so the stationary or long term temperature differ from the currently measured signal. In order to prevent critical temperature, which could damage the engine, it is necessary to describe the behavior of retarded observables. Furthermore, highly retarded and critical observables, e.g. temperatures, have to be extrapolated in order to interrupt measurement ramps before any damage can happen to the system. So, we have to extrapolate the temporal trend of critical sensor values. This cannot be done by linear regressions in a proper way, therefore a specific fitting function for retarded signals is applied to the measurement data (Equation (65)). The fitting parameters  $A$ – $E$  are calculated with the least-squares method. If the parameters have been calculated, it is possible to extrapolate the critical observable.

$$f(x) = A + B^{C \cdot (x-D)} \cdot \exp(E) \quad (65)$$

In this work, we integrated this method for the prediction of the critical temperatures. Therefore, we compute the fitting parameters, if a threshold temperature has been passed. This threshold temperature is lower than the hard boundary temperature. The range of the prediction is configured by the user, typically one extrapolates the critical temperature in a temporal range of 10 to 60 seconds.

#### 4.2.4.5 Detection of driveable Data Space

It is possible to damage the combustion engine during the measurement process. A subset of actuator combinations causes critical cylinder pressures, catalyst temperatures, etc.. Depending on the measurement technique, the physical system is varied with different actuator gradients. That is why the most measurement programs try to detect the borders of the operational data space. In case of *AmmOC* the variation speed is rather low, i.e. full range of single actuator in passed in 10 minutes, so the observables can be monitored directly. If the measurement ramp enters an unstable region, *AmmOC* has enough time to interrupt the measurement ramp and continue with the new operation point.

---

### 4.2.5 Measurement Specialization

---

In order to solve an optimization problem we have to vary all relevant actuators. Typically, the combinations of actuators for ideal objective values are hard to find. That is why the search for ideal points starts with a coarse resolution and ends with a very fine one. *AmmOC* contains two different refinement techniques, the refinement due to unexpected result values and the refinement due to stationary solution improvements. Both refinement techniques modify the data tree, which have been defined in Subsection 4.2.1.

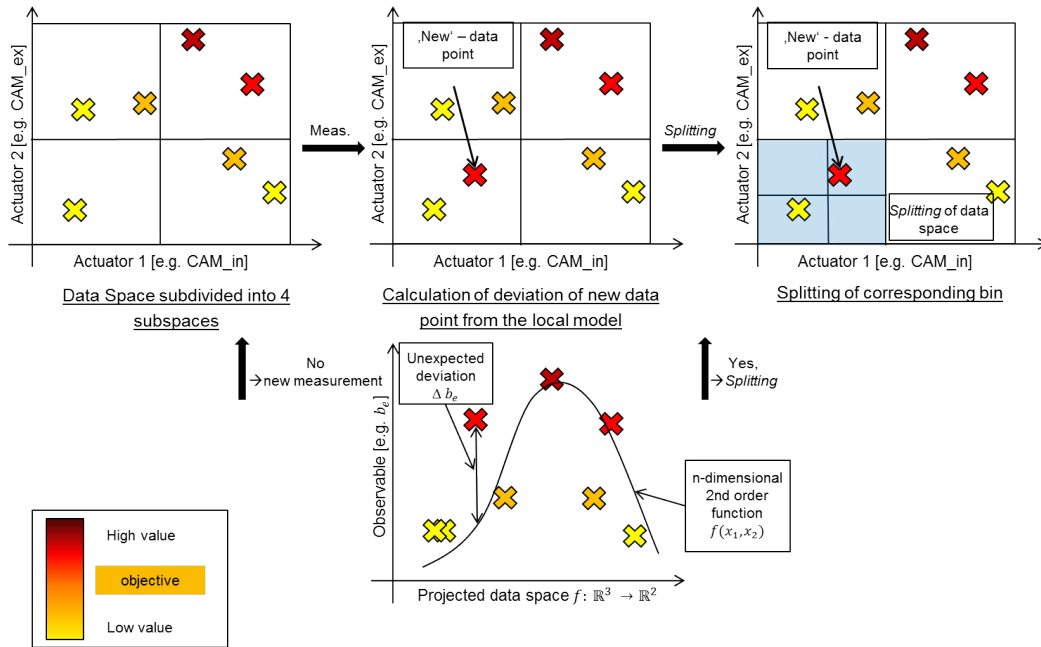
#### 4.2.5.1 Refinement due to unexpected Result Value

In this mode, the algorithm refines the data tree, if a measured observable of interest is higher or lower than expected. Therefore, the algorithm performs local fits, with a 2nd order polynomial function in  $n$  dimensions.  $n$  is given by the number of *dynamic actuators*. New



data points  $d_i \in D$  can be compared to a corresponding fitted value  $\hat{d}_i$ . The fitted value  $\hat{d}_i$  is determined with the fitted function at the position of  $d_i$ . The valid data points in the immediate vicinity of  $d_i$  contribute to the evaluation. The minimal number of data points for the evaluation of the fitting parameter is given by  $10 \cdot (n + 1)^2$ . In the first instance, one adds all nearest neighbors of  $d_i$  to the set of data points. If the number of data points is smaller than the minimal number, one continues to add the nearest neighbors of the nearest neighbors until the minimal number has been overcome. The determination of nearest neighbors of  $d_i$  is explained in Subsection 4.4.1. The fit function is derived with the least-squares method.

This process is illustrated in Figure 41. Before the new measurement cycle, the data tree contains four bins, so the whole data space is subdivided into four quadrants. A new data point  $d_i$  is measured in the lower-left quadrant. So, the algorithm performs a 2nd order polynomial fit, with two dimensions, i.e.  $CAM\_in$  and  $CAM\_ex$ . The evaluation of the fit shows that the newly measured value has an unexpected deviation from the local model, thus the algorithm performs a split in the lower-left quadrant.



**Figure 41:** Illustration of measurement specialization due to unexpected derivations of observables.

The consequences of the splitting process will be explained in Subsection 4.2.5.3.

#### 4.2.5.2 Refinement due to Improvement of stationary Solution

The second criterion for measurement refinement is coupled to improvements of the unconstrained solution, which is given by the best values of each bin of the solution tree *Sol\_Tree*. This data tree has been introduced in Subsection 4.2.1. New valid data points are compared to the current best entries in *Sol\_Tree*. Therefore, the new valid data point is sorted into the data tree. If the new valid data point contains a better specific fuel consumption value or more ideal emission values, this module initializes a measurement refinement. In general, observables are classified into *ignore*, *hard*, *hard integral* and *soft*. If the classification of an observable is unequal *ignore*, the method creates an accordant unconstrained solution, in order to store the most ideal of each observable of interest.

In contrast to the first refinement method, this method includes one more analysis step before the splitting procedure. After the method detects a valid data point, which improves the unconstrained solution of any criterion, the refinement routine analyses, which actuator has caused the improvement. Typically, the algorithm varies more than one actuator, while detecting a solution improvement, that is why the actuator with the biggest impact on the optimization criteria is not obvious. After the detection process, the method adds additional measurement ramps to the measurement plan. These new measurement passages contain single actuator variations. This measurement part is called *cross measurement*. Afterwards, the algorithm compares the best values of each stroke of the *cross measurement* in relation to the criterion, that initiated the solution improvement. Since only one actuator is varied during one stroke, it is possible to identify the actuator with the biggest impact on the solution improvement. This information defines the asymmetric split of the corresponding bin of the data tree. The splitting process will be explained in the following Paragraph 4.2.5.3.

#### 4.2.5.3 Splitting Process

**Symmetric Split Method:** *AmmOC* distinguish between symmetric and asymmetric bin splits in data trees (see Subsection 4.2.1). A symmetric split cuts each actuator range in a bin into two equal parts. An asymmetric split cuts only a subset of the defined dynamic actuator ranges into two equal parts in a selected bin.

If the sensor values of a new data point have a larger deviation than a user defined threshold from the local polynomial model (see Subsection 4.2.5.1) or if the new data point improves the stationary solution (see Subsection 4.2.5.2) the corresponding bin will be split. Furthermore, the user can define a maximal depth of the data tree. Depending on the criterion that has been met, the bin will be split symmetrically or asymmetrically.

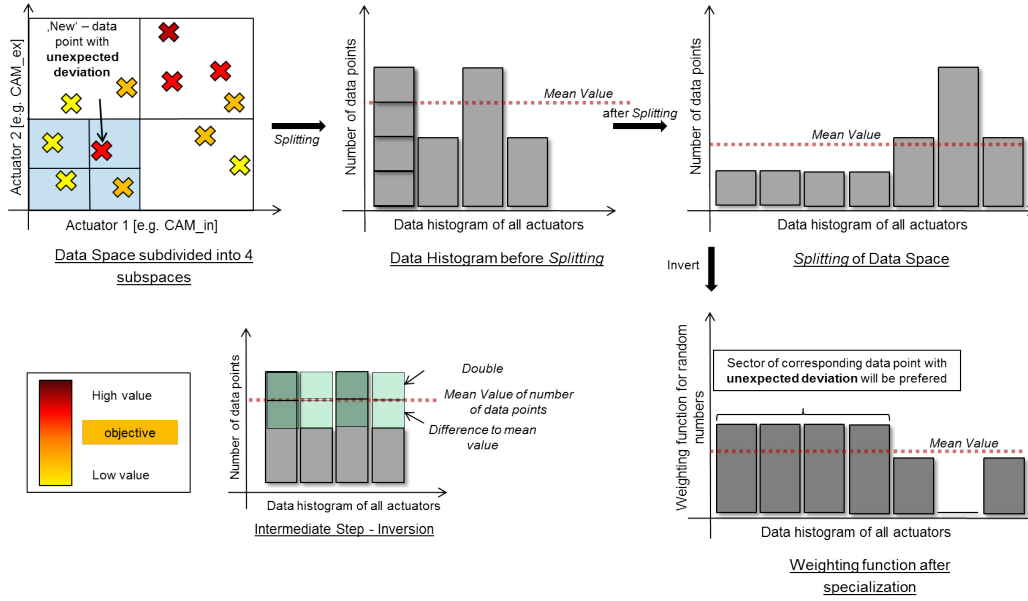
After the split is done, all valid data points of the selected bin have to be sorted into the newly split bins. Consequently, the data density function is modified, the number of bins increases and the number of events in each new bin is smaller than in the original bin. That is why the weighting function for the random numbers increases in this specific region of the data space. Therefore, the probability for a new measurement ramp that ends in the newly discovered region increases, too (see Figure 42).

**Asymmetric Split Method:** This method is an upgrade of the *measurement specialization*. The purpose of this method is to increase the measurement resolution at a specific location in the data space by splitting the data tree. But in contrast to the symmetric split this method analyses the system behavior, in order to split the data tree more specifically.

This method splits the data tree not for each dynamic actuator, but in a subset of dynamic actuators  $\bar{A}$ , which caused the observed systematic behavior of observable  $s$ . Therefore, the algorithm has to find out, which actuators  $\bar{A}$  have caused the improvement of the best entry in *Sol\_Tree* in the current bin  $sol_k$ . As already introduced, the algorithm performs *cross measurements* of all actuators  $j \in A$  within the current ranges of the bin  $sol_k \in Sol\_Tree$ . So, we compare the smallest observed sensor value  $\underline{sol}_k^j$  of observable  $j$  to the measured data points in the *cross measurement*. So, we derive a maximal improvement

$$I_{max} = |\underline{sol}_k^j - d_i^j|$$

where  $i$  goes from the begin till the end of the *cross measurement* and  $j$  is the sensor value, which caused the refinement process. This maximal improvement  $I_{max}$  has to be compared to the improvements within the single 1-dimensional measurement ramps in the *cross measurement*. During a single 1-dimensional ramp the algorithm varies one actuator  $k$ , only. So,



**Figure 42:** Illustration of splitting process on the data tree and recalculation of the weighting function.

the algorithm observes the maximal improvement of the selected observable and actuator during a single ramp  $I_{max}^k$ . Finally, the specific relevance of actuator  $r_k$  is given by  $r_k = \frac{I_{max}^k}{I_{max}}$ . The user defines a lower bound  $\underline{r}_k$ , if the specific relevance  $r_k \geq \underline{r}_k$  the algorithm assumes, that the actuator  $k$  contributes to the improvement and adds  $k$  to  $\bar{A}$ .

#### 4.2.6 Virtual Test Bench

The virtual test bench is supposed to simulate an internal combustion car engine. This module has been developed in cooperation with P. Lind and includes the work of G. Schreiber et al., R. Golloch et al., J. Warnatz et al. and E. Köhler et al. Primary, the task of virtual test bench is to deliver data for testing the measurement and optimization algorithms of *AmmOC*. A high degree of realism of this approach is not intended. The input variables of this model are given by the static  $A_s$  and dynamic  $A_d$  actuators of  $d_i \in D$  (see Definition (2)). Furthermore, the model includes physical constants and model parameters. The output of this model is given by the sensor values  $M$  of data point  $d_i$ .

In this work, we optimize the actuator settings of two models, the first model has been contributed by AVL and is on the basis of measured data, the second one will be described in the following. The motivation for the introduction of the second model is given by the higher complexity of the own virtual test bench model. This model depends on four dynamic actuators, which have severe impact on the *Air path* (see Subsection 2.2.1.1). The AVL engine model has an easier *Air path* system.

The empirical model assumes two inner cylinder zones, an unburnt and a burnt one. The unburnt zone is given by temperature  $T_1$ , pressure  $p_1$  and volume  $V_1$ . The burnt zone is given by  $T_2$ ,  $p_2$  and  $V_2$ . The initial conditions of this model are varied by the engine variables, the opening angle of the throttle valve  $\alpha$ , the opening angle of the exhaust gas recirculation (EGR) valve  $\alpha_{EGR}$ , the revolution frequency of the crankshaft  $rpm$ , the ignition timing  $\alpha_{spark}$ , the phase shift of intake and exhaust cam shaft  $CAM_{in}$  and  $CAM_{ex}$  and the

fraction of air to fuel that takes part in the combustion  $\lambda$ . The description of these actuators is given in Subsection 2.2.1.1. The observables, which we want to compute are introduced in Subsection 2.2.2.1.

First of all, we have to determine the function  $V_1(^{\circ}\text{CA})$  of the volume of the cylinder  $V_1$ , where  $^{\circ}\text{CA}$  is the current rotation angle of the crank train. This current volume between piston and dome is defined by the length of the piston rod  $l_{\text{piston\_rod}}$ , the lift of the crankshaft  $P_{\text{crankshaft}}$ , the diameter of the engine's bore  $d_{\text{bore}}$  and the volume of the cylinder dome  $V_{\text{dome}}$ . The volume is given by:

$$V_1(^{\circ}\text{CA}) = (l_{\text{piston\_rod}} + P_{\text{crankshaft}} - (P_{\text{crankshaft}} \cdot \cos(^{\circ}\text{CA} = 180^{\circ})) + \sqrt{l_{\text{piston\_rod}}^2 - P_{\text{crankshaft}}^2 \cdot \sin(^{\circ}\text{CA} = 180^{\circ})^2} \cdot \pi \cdot \frac{d_{\text{bore}}^2}{2}) + V_{\text{dome}} \quad (66)$$

The lift functions  $P_{\text{CAM}_{\text{ex/in}}}$  of the intake and exhaust valves depend on the half width of the exhaust/intake valve lift function  $w_{\text{CAM}_{\text{ex/in}}}$ , the height of the exhaust/intake valve lift function  $h_{\text{CAM}_{\text{ex/in}}}$ , the center position of the intake/exhaust cam  $\text{CAM}_{\text{ex/in}}^0$ . The lift function is given by:

$$P_{\text{CAM}_{\text{ex/in}}} = \frac{1}{w_{\text{CAM}_{\text{ex/in}}} \cdot \sqrt{2} \cdot \pi} \cdot h_{\text{CAM}_{\text{ex/in}}} \cdot \exp\left(-\frac{1}{2} \cdot \left(\frac{\text{CAM}_{\text{ex/in}}^0 + \text{CAM}_{\text{ex/in}}}{w_{\text{CAM}_{\text{ex/in}}}}\right)^2\right) \quad (67)$$

The function of volume  $V_2(^{\circ}\text{CA})$  is associated with the burnt zone. It describes the volume gain due to the opening exhaust valves. The effective volume of the manifold  $V_{\text{manifold}}$  is added to  $V_1$  by the opening cross section of the exhaust valves.

$$V_2(^{\circ}\text{CA}) = V_1 + \int_{P_0 + \text{CAM}_{\text{ex}} + \text{CAM}_{\text{ex}} - 2 \cdot w_{\text{CAM}_{\text{ex}}}}^{\text{CA}} P_{\text{CAM}_{\text{ex}}} (^{\circ}\text{CA}') \cdot d^{\circ}\text{CA}' \cdot V_{\text{manifold}} \quad (68)$$

A very important parameter of 4-stroke engines is valve overlap  $A_{\text{overlap}}^{\text{valve}}$  of the valve lift functions  $P_{\text{CAM}_{\text{ex/in}}}$ . Typically, this is a critical and very important engine parameter. The primary use of this setting is the compliance of emission regulations, since the fraction of unburnt particles is reduced.

$$A_{\text{overlap}}^{\text{valve}} = (\alpha_{\text{EGR}} + 1) \cdot \int_{600}^{800} \min(P_{\text{CAM}_{\text{ex}}} (^{\circ}\text{CA}), P_{\text{CAM}_{\text{in}}} (^{\circ}\text{CA})) d^{\circ}\text{CA} \quad (69)$$

The valve overlap  $A_{\text{overlap}}^{\text{valve}}$  has an effect on the temperature  $T_1$  of the unburnt zone, too. The recirculation of burnt gas causes a mixing process of burnt ( $T_2$ ) and unburnt ( $T_1$ ) gas in the cylinder. The initial mixing temperatures are the model parameters  $T_1^{\text{mix}}$  and  $T_2^{\text{mix}}$ .

$$T_1 = A_{\text{overlap}}^{\text{valve}} \cdot T_2^{\text{mix}} + \alpha \cdot (1 - A_{\text{overlap}}^{\text{valve}}) \cdot T_1^{\text{mix}} \quad (70)$$

The next passage focuses on the air intake of the model. Generally, the piston moves down and soaks in the air, which it needs for the combustion of the injected fuel. The air intake

process depends on the opening cross section of the system and its flow resistance. Flow resistance is mainly generated by valves, filters and the exhaust back pressure. Furthermore, the air filling rate depends on the inertia of the air mass, therefore the revolution frequency of the crankshaft, and thus the velocity profile of the pistons are crucial for the amount of the consumed air. This effect can be modeled by a delayed crank angle  $^{\circ}\text{CA}'$  which effects the air filling rate function  $r_{Air}(^{\circ}\text{CA})$ . The parameters  $a, b, c, d$  are fit coefficients from a fit of the model to realistic data. Their values are  $a_{inertia} = 4e-14$ ,  $b_{inertia} = 8e-12$ ,  $c_{inertia} = 2e-7$ ,  $d_{inertia} = 6e-4$ . So, we obtain:

$$^{\circ}\text{CA}' = ^{\circ}\text{CA} - (a_{inertia} \cdot rpm^4 + b_{inertia} \cdot rpm^3) + c_{inertia} \cdot rpm^2 + d_{inertia} \cdot rpm \quad (71)$$

The function  $r_{Air}(^{\circ}\text{CA})$  is divided into four phases:

$$r_{Air}(^{\circ}\text{CA}) = \begin{cases} P_{CAM\_ex}(^{\circ}\text{CA}) \cdot V_1(^{\circ}\text{CA}) & \text{if } ^{\circ}\text{CA} \in (720, 810) \\ 0 & \text{if } ^{\circ}\text{CA} \in (540, 720] \\ P_{CAM\_in}(^{\circ}\text{CA}) \cdot V_1(^{\circ}\text{CA}) & \text{if } ^{\circ}\text{CA}' \in (640, 900) \wedge ^{\circ}\text{CA} > 720 \\ P_{CAM\_in}(^{\circ}\text{CA}) \cdot V_1(^{\circ}\text{CA}) & \text{if } ^{\circ}\text{CA}' \in (640, 900) \wedge ^{\circ}\text{CA}' < 280 \\ -P_{CAM\_in}(^{\circ}\text{CA}) \cdot V_1(^{\circ}\text{CA}') & \text{else} \end{cases} \quad (72)$$

The total mass of air for the combustion  $Air\_fill$  is given by the opening angle of the throttle valve  $\alpha$  and the integral of  $r_{Air}(^{\circ}\text{CA})$ .

$$Air\_fill = \alpha \cdot \int_{180}^{900} r_{Air}(^{\circ}\text{CA}) d^{\circ}\text{CA} \quad (73)$$

Furthermore, the behavior is fitted to real data for different settings of the throttle valve  $\alpha$  by:

$$Air\_fill\_scale(\alpha, rpm, CAM_{in}) = Air\_fill(\alpha, rpm, CAM_{in}) \cdot (-8.7533 \cdot \alpha^3 + 13.943 \cdot \alpha^2 - 4.6344 \cdot \alpha + 0.4063) \quad (74)$$

After the intake period of the engine has finished and all valves are closed, the next phase starts. The compression phase leads to a pressure rise from the initial pressure of the unburnt zone  $p_1^{init}$  to

$$p_2^{pre} = p_1^{init} \cdot \frac{V_1^{\kappa}}{V_2} \quad (75)$$

and a temperature rise

$$T_2^{pre} = T_1(^{\circ}\text{CA} = \alpha_{spark}) \cdot \frac{V_1^{\kappa-1}}{V_2} \quad (76)$$

where  $\kappa$  is the adiabatic coefficient. This process is assumed to happen nearly adiabatic. In the last part of the compression, the ignition process of the combustible mixture is initialized by a electric spark. This ignition spark causes a chain reactions of chemical radicals that

convert the fuel-air mixture into energy. An important fact is, that the ignition process happens not instantaneously. So, we have to determine the ignition delay  $\tau_{ign\_delay}$ . Basically this equation depends on the air ration  $\lambda$  and the compression ratio, which lead to a specific compression pressure  $p_1$  ( $^{\circ}\text{CA} = \alpha_{ign}$ ) and compression temperature  $T_1$  ( $^{\circ}\text{CA} = \alpha_{ign}$ ).

$$\alpha_{ign\_delay} = 0.0187 \cdot 0.95^3 \cdot 402 \cdot (1 + 0.96 \cdot ((\frac{1}{1.1} \cdot \lambda) - 1) + 5.45 \cdot ((\frac{1}{1.1} \cdot \lambda) - 1)^2 + 8.32 \cdot ((\frac{1}{1.1} \cdot \lambda) - 1)^3) \quad (77)$$

$$\tau_{ign\_delay} = \frac{1}{1000} \cdot \alpha_{ign\_delay} \cdot p_1(^{\circ}\text{CA} = \alpha_{ign})^{1.7} \cdot \exp(\frac{3800}{T_1(^{\circ}\text{CA} = \alpha_{ign})})^{34} \quad (78)$$

This simple model works with an symmetric conversion rate  $Conv(^{\circ}\text{CA})$  of the fuel. This conversion rate contains an additional modeling parameter  $f_{HC}$  that describes the fraction of unburned carbon hydrides of the injected fuel. So, we obtain:

$$Conv(^{\circ}\text{CA}) = \int_{\alpha_{spark}}^{\alpha_{spark} + \alpha_{conv}} \frac{1 - f_{HC}}{\sigma_{Conv} \cdot \sqrt{2 \cdot \pi}} \cdot \exp(-0.5 \cdot (\frac{\alpha_{spark} + \alpha_{conv} - \alpha_{spark} - \sigma_{conv}}{\sigma_{conv}})^2) d\alpha_{conv} \quad (79)$$

where  $\sigma_{Conv}$  is a fit parameter of the system, which describes the conversion speed of the fuel.

The conversion of fuel lead to temperature rise  $T_2^{post} (^{\circ}\text{CA})$  in the burnt zone of the two zones model. Furthermore, the current temperature of the burnt zone depends on the temperature of the previous calculation time step  $T_2^{post} (^{\circ}\text{CA} - \alpha_{resolution})$  where  $\alpha_{resolution}$  is the resolution of pressure and temperature functions in the model. A non negligible energy loss is given by the heat transfer of the burnt zone to the exposed cylinder walls. The area of exposed wall depends on the geometry of the crank mechanism and is given by:

$$A_{cylinderwall} (^{\circ}\text{CA}) = 2 \cdot \pi \cdot \frac{d_{bore}}{2} \cdot (l_{piston\_rod} + P_{crankshaft} - (P_{crankshaft} \cdot \cos(^{\circ}\text{CA})) + \sqrt{l_{piston\_rod}^2 - P_{crankshaft}^2 \cdot \sin(^{\circ}\text{CA})^2}) \quad (80)$$

A important output value of this model is the critical component temperature  $T_{crit}$ . It describes the maximal temperature of the exhaust gas, when the exhaust valve opens. This temperature is a hard limit for the most combustion engines, in order to prevent thermal stress for the exhaust turbochargers or catalytic converter, which are mounted close the the exhaust manifold. The critical temperature is defined by  $T_2^{post} (^{\circ}\text{CA})$  at the moment, when the exhaust valves opens. It is given by:

$$T_2^{post} (^{\circ}\text{CA}) = T_2^{post} (^{\circ}\text{CA} - \alpha_{resolution}) - Conv(^{\circ}\text{CA}) \cdot A_{cylinderwall} (^{\circ}\text{CA}) \cdot T_{scale} \cdot (T_1 - T_2^{post} (^{\circ}\text{CA} - \alpha_{resolution}))^2 \quad (81)$$

<sup>34</sup> These equations have been taken from [Sch06]

where  $T_{scale}$  is a fit parameter of the system ( $T_{scale} = 1E - 9$ ). It is used to scale thermal losses at the cylinder walls.

$$T_{crit} = T_2^{post}(\text{°CA} = CAM_{ex}^0 + CAM_{ex} - 2 \cdot w_{CAM_{ex}}) \quad (82)$$

The temperature rise of the burnt zone influences the pressure of the burnt zone  $p_2^{post}$ . Generally,  $p_2^{post}$  describes the interaction of compression, valve overlap and burnt zone temperature. The critical pressure  $p_{crit}$  is given by the maximal value of  $p_2^{post}(\text{°CA})$ . Like  $T_{crit}$   $p_{crit}$  is a hard boundary of the combustion engine. So, we define  $p_2^{post}$ :

$$\lambda_L = 1 - \frac{A_{overlap}^{valve}}{10 \cdot 100} \quad (83)$$

$$p_2^{post}(\text{°CA}) = p_1 \cdot \frac{V_1}{V_2(\text{°CA})} + \lambda_L(A_{overlap}^{valve}) \cdot scale \cdot Air\_fill\_scale(\alpha, rpm, CAM_{in}) \cdot Conversion(\text{°CA}) \cdot p_1 \cdot \frac{V_1}{V_2(\text{°CA})} \cdot \frac{T_1}{T_2(\text{°CA})} \quad (84)$$

where  $scale$  is a model parameter to scale pressure increase due to combustion. The critical pressure is given by the maximum of the pressure function

$$p_{crit} = \max(p_2^{post}) \quad (85)$$

This model does not threat knocking or pre-ignited combustion.

The fundamental equation for this model have been derived. Now, we are able to compute the requested sensor values of the engine. The mass of air  $m_L$  of an engine with a defined displacement  $V_{displacement}$  is given by:

$$m_L = \lambda_L * Air\_fill \cdot p_1 * \frac{V_{displacement} + V_{dome}}{287.058 * T_1}^{35} \quad (86)$$

Consequently, the mass of injected fuel  $B$  is defined by  $m_L$  and air-fuel ratio  $\lambda$ :

$$B = \frac{m_L}{\lambda \cdot 14.7} * \frac{3600}{CA2seconds(720, rpm)}^{36} \quad (87)$$

where  $CA2seconds(720, rpm)$  is a function that calculates the time in seconds for passing 720 °CA at the current revolution frequency  $rpm$ . In order to compute the resulting engine torque  $M_{avg}$ , we have to determine the force  $F_k(\text{°CA})$  on the piston:

$$F_k(\text{°CA}) = p_2^{post}(\text{°CA}) * \pi * \frac{d_{bore}^2}{4}^{37} \quad (88)$$

<sup>35</sup> This equation has been taken from [Gol05]

<sup>36</sup> This equation has been taken from [Gol05]

The mean engine torque  $M_{avg}$  is given by the following integral:

$$M_{avg} = \int_{180}^{900} F_k(^{\circ}\text{CA}) \cdot P_{crankshaft} * (\sin(^{\circ}\text{CA}) + \frac{\frac{P_{crankshaft}}{l_{piston\_rod}} P_{crankshaft} \cdot \sin(2 \cdot ^{\circ}\text{CA})}{2 \cdot \sqrt{1 - \frac{P_{crankshaft}^2}{l_{piston\_rod}^2} \cdot \sin(^{\circ}\text{CA})^2}}) d(^{\circ}\text{CA})^{38} \quad (89)$$

Finally, the specific fuel consumption  $b_e$  is given by:

$$b_e = \frac{B}{M_{avg} * \frac{rpm}{9550}}^{39} \quad (90)$$

<sup>37</sup> This equation has been taken from [Koe06]

<sup>38</sup> This equation has been taken from [Koe06]

<sup>39</sup> This equation has been taken from [Gol05]



---

## 4.3 Data Preparation and Compression

---

### 4.3.1 Time Series Compressor

---

After the raw data points have been processed by the moving linear regression method (see Subsection 4.2.4.3), we have to reduce the number of data points in order to improve the run-time of the following optimization algorithms. The time series compressor *TSC* applies two criteria on the raw data points. The first criterion is given by a minimal recording frequency  $\nu_{min}$ . Generally, the measurement frequency  $\nu_{meas}$  is given by the data acquisition system, e.g. 5 to 100 Hz. The second criterion is the maximal allowed relative change  $\delta_{max}^m$  of sensor  $m \in M$ , which contribute to the objective function or to the constraints. So, the *TSC* will convert more raw data points to valid data points, if important sensor values, e.g. specific fuel consumption changes quickly. If important the sensor values do not change quickly, the number of valid data points in this time window will be small. The effective measurement frequency  $\bar{\nu}$  is given as the inverse time difference from the current data point  $t(d_i)$  and the last stored data point  $t(d_{i-1})$ .

$$\bar{\nu} = \frac{1}{t(d_i) - t(d_{i-1})} \quad (91)$$

So, the criteria can be written as:

$$\text{Store if} = \begin{cases} \bar{\nu} \leq \nu_{min} \\ |d_i^m - d_{i-1}^m| \geq \delta_{max}^m \forall m \in M \end{cases} \quad (92)$$

The set of compressed data points is a subset of the raw data points  $D$ . Each data point  $d_i \in D$  has an according status value  $c_i \in C$  which describes the current status of the data point. The first data point of a ramp has status 1. If the minimal relative deviation has been archived the status is set to 2. If the lower bound of the measurement frequency has been reached, the status is set to 3. If a data point has to be compressed by *TSC* the status is set to 0, consequently, this raw data point will not take part in the following analysis steps. A status value greater than 3 is caused by the adaptive space compressor, which will be explained in the following paragraph.

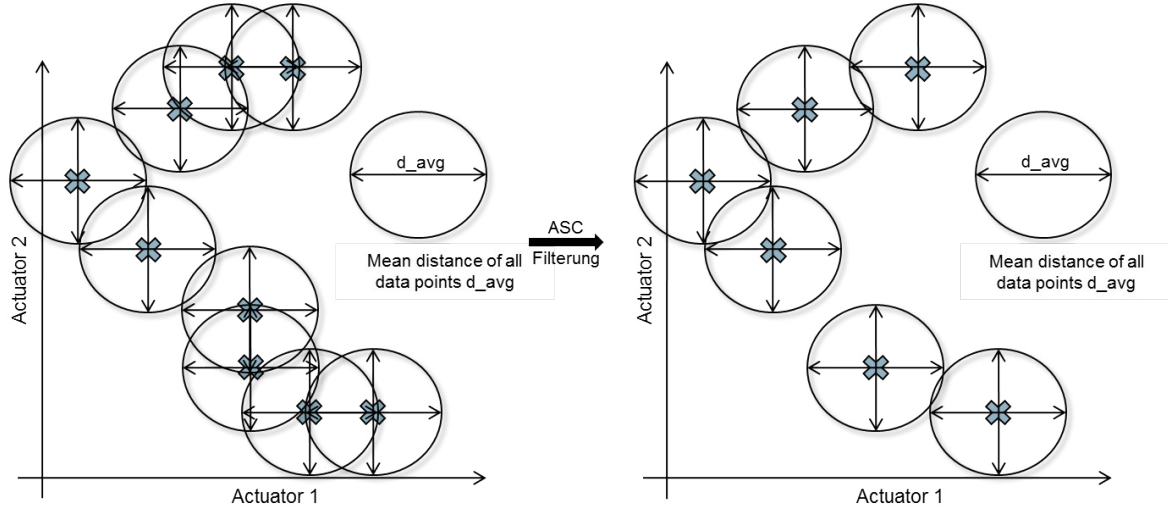
---

### 4.3.2 Adaptive Space Compressor

---

The adaptive space compressor *ASC* algorithm has been established, in order to reduce the number of data points. In contrast to the *TSC* (see Subsection 4.3.1), the *ASC* filters the data points with respect to the spatial distribution of the data points. Due to the measurement method, which is characterized by measurement ramps, the spatial distribution contains local inhomogeneities. Our analysis routines, e.g. Voronoi tessellation and local models, benefit from a spacial data homogeneity. Thus, we have to define an algorithm that reduces the number data points, but preserves the optimization information. Additionally, *ASC* is used to perform validation checks on previously measured data streams. The algorithm will be explained in the following.

In General, this method analyses the data tree *Tree*, which contains the fragmentation of the data space and its bin occupancy. The *ASC* filters data points, which have not been filtered by *TSC*, in order to increase homogeneity of the data point distribution. First, we have to



**Figure 43:** Illustration of Adaptive Space Compressor method

compute the mean distance  $\bar{d}$  of all data point  $d_i \in D$  in accordance with its actuators  $k \in A$  among each other:

$$\text{dist}_A(d_i, d_j) = \sqrt{\sum_{k \in A} (d_i^k - d_j^k)^2} \quad (93)$$

$$\bar{d} = \frac{1}{n} \cdot \sum_{i=0}^n \cdot \frac{1}{n-1} \cdot \sum_{j=0 \wedge j \neq i}^{n-1} \text{dist}_A(d_i, d_j) \quad (94)$$

The ASC method is illustrated in Figure 43. On the left, we see the original data point distribution and the mean distance environment of each data point. On the right, we see the resulting distribution. The analytic steps of the algorithm are described in the following.

The mean distance  $\bar{d}$  is used as general length scale parameter of the current data point distribution. Furthermore, the split depth of the data tree and the structure of the corresponding bin is taken into account. Due to refinement processes, the algorithm has performed several symmetric and asymmetric splits on the data tree  $T$ . The width  $\delta_b^k$  of the corresponding bin  $t_k \in T$  of data point  $d_i$  influences the filter process. So, the algorithm checks if one point  $d_i$  is within the environment of any other point  $d_j$ . This is the case if

$$\text{dist}_A(d_i, d_j) \leq \bar{d} \cdot s_{ASC} \cdot \sqrt{\sum_{k \in A} \delta_b^k \cdot (d_i^k - d_j^k)^2} \quad (95)$$

where  $s_{ASC}$  is the ASC length scale (User parameter).

The result is saved as triangle matrix  $D_{\text{conflict}} \in \mathbb{R}^{n \times n}$  where  $n$  is the number of data points. A short example is given in Table 5. The frequency of conflicted combinations  $H \in \mathbb{Z}^n$  of  $D_{\text{conflict}}$  is calculated by counting the 1s in the selected column and row of  $D_{\text{conflict}}$ . Data points are compressed by ASC by the following algorithm:

1. First we have to get the biggest entry  $i$  in  $H$ . This is equal to the selection of the data point  $d_i$ , which causes the biggest number of conflicts in  $D_{\text{conflict}}$ .

**Table 5: Example for  $D_{conflict}$**

	0	1	2	3	4	5	6
0							
1	1						
2	0	1					
3	1	1	1				
4	0	0	0	1			
5	0	1	0	1	0		
6	0	1	0	1	0	0	
7	1	1	1	1	1	1	0

2. If index  $i$  is valid:

- Compress point  $d_i$ . So we mark data point  $d_i$  as compressed by changing the status of  $d_i$ .
- Afterwards, we set the corresponding entry in  $H[i] = 0$ , where  $H[i]$  is entry  $i$  of array  $H$ .
- Loop through line  $i$  of  $D_{conflict}$  with  $j$ :  
 $H[j] = H[j] - 1$  if  $D_{conflict} = 1$
- and loop through column  $i$  of  $D_{conflict}$  with  $j$ :

$H[j] = H[j] - 1$  if  $D_{conflict} = 1$ . In the previous two steps we decrease the number of conflicts in  $H[j]$  by the number of conflicts, which have been caused by data point  $d_i$ .

3. Else:

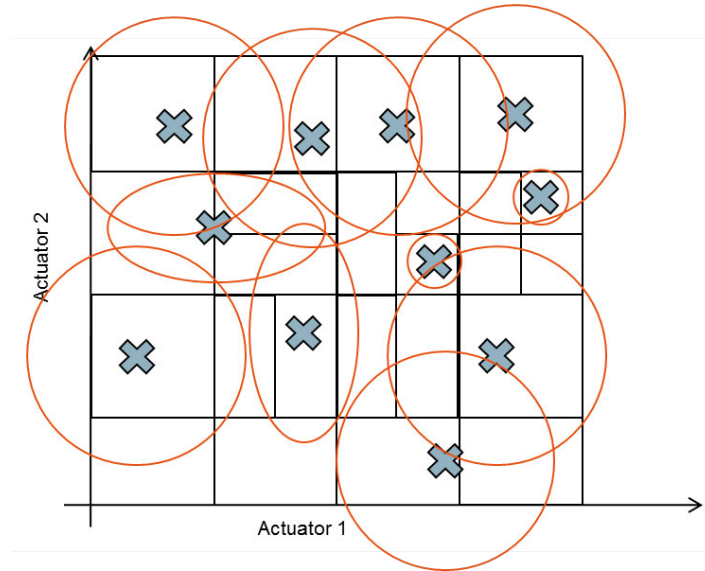
Finish

The previous example does not include the specific structure of the data tree  $T$ . Depending on the splitting processes on  $T$ , the bins cover different amount of areas, since the width of each bin is getting smaller with each split (see Subsection 4.2.5). A bin is split, in order to rise the importance of this region. If a bin is split, one region is no longer represented by one bin, but by  $2^d$  bins. That is why a spacial filter algorithm has to take this information into account. Consequently, the hyper sphere becomes a hyper ellipsoid (see Figure 44). The conflicted combinations  $H$  of  $D_{conflict}$  have to be determined with the new sphere definition. Afterwards, the mechanism of ASC can be applied as usual.

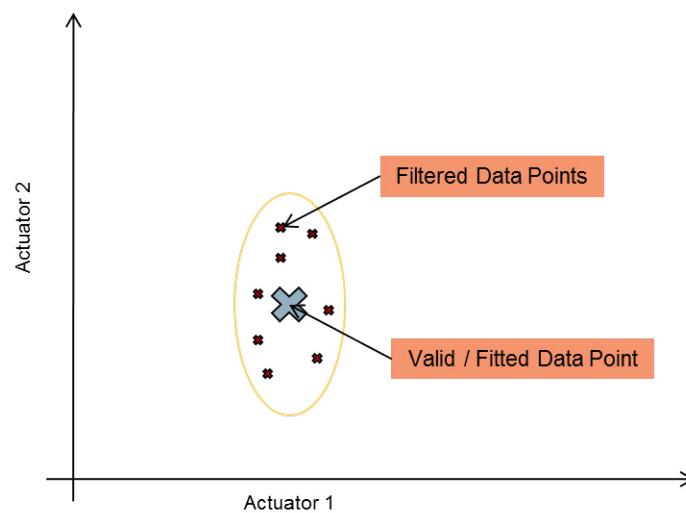
Finally, the ASC algorithm performs plausibility checks for newly measured data points. Therefore, ASC calculates the standard deviation  $\sigma_j$  of all data points  $d_i$ , which are within the monitored ASC hyper ellipsoid:

$$\sigma_j = \sqrt{\frac{(\bar{d}_i^j - d_i^j)^2}{n_k}} \quad (96)$$

where  $\bar{d}_i^j$  is the average value of the observable of interest  $j$  of data point  $d_i$  and  $n_k$  is the number of data points in the monitored hyper ellipsoid  $k$ . It is essential, that an ASC hyper ellipsoid represents all data points (see Figure 45) that are member of the hyper ellipsoid,



**Figure 44:** Hyper ellipsoid (red) for representing data tree structure (black) in ASC method.



**Figure 45:** Illustration of raw and valid data points within a hyper ellipsoid in the ASC method.

---

otherwise the filter process would cause an incorrect information. Two reasons are possible for such incorrect information, the size of the ASC ellipsoid is too large and should be scaled down or the systematic behavior of the engine has changed. The second one occurs, if a physical damage happen to the experimental setup. This can be easily detected, because the systematical behavior happens spontaneously, therefore the algorithm is able to measure the undamaged system behavior, which can be compared to the newly measured damaged system behavior. In this case, the user is informed. A newly measured data point is defined as not ellipsoid conform, if the measured value of an observable of interest is outside the  $2\sigma$  environment of the average value of the ASC hyper ellipsoid.

In case of filtered data points, the algorithm has to choose, which point has to be filtered within the hyper ellipsoid. This decision is based on the user definition. The user chooses between the mean value of all participants or the data point with the best value in the selected criterion. Due to the fact, that the upper limit of nitrogen oxides are the most critical criterion, one configures the selection mode to pick the data lines, that provide the best values for nitrogen oxides.

---

## 4.4 Data Analysis and Discretization

---

### 4.4.1 Determination of nearest Neighbors

---

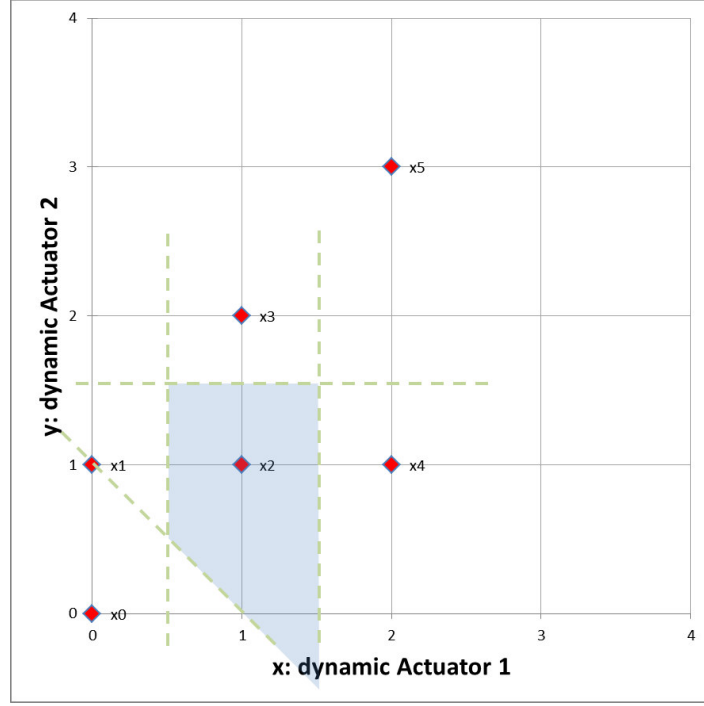
The nearest neighbor information is essential for the classification of multi-dimensional data. The local indexing helps to create paths through the data point cloud and helps to determine n-dimensional data fits. The data classification is done by Voronoi diagrams. The theoretical definition of Voronoi diagrams is given in Subsection 2.1.5. An established algorithm is explained in Subsection 2.3.2. The next passage explains the basic idea of an own implementation of a Voronoi tessellation algorithm. This covers a basic idea, an initial and an iterative mode of the algorithm. This algorithm is based on the native idea of *Voronoi* tessellation by determine all corresponding vertices of the *Voronoi* cell. The massive parallel implementation on graphical processing units *GPUs* increases the performance of the algorithm, so that it is able to compete with state-of-the-art methods. It has been necessary to introduce our own algorithm that determines the nearest neighbors of our data points, since the established algorithm have longer runtimes. This aspect will be investigated in the result section (see Subsection 5.2.1). Furthermore, we validate the new algorithm with the help of the established and proven *Beneath-and-Beyond* method of *Polymake* [Jos02].

#### 4.4.1.1 Mathematical Description of the Recursive Light Ray Algorithm

The mathematical description of this algorithm is supported by an introductory example. In this two dimensional example (see Figure 46), we want to determine the nearest neighbors of data point  $x_2$  at the location (1,1). The blue filling represents the expansion of the Voronoi cell, which we have to compute. The green lines represent the perpendicular bisector planes between the corresponding points. We introduce  $PBP(x_2, x_1)$  as abbreviation for the perpendicular bisector plane of data point  $x_2$  and  $x_1$ .

The first step of the computation of the Voronoi cells is given by the determination of all PBPs from  $x_2$  to all other data points. This will be abbreviated with  $PBP(x_2, \cdot)$ . The general form of the equation of a plane  $E$  is given by

$$E : (\vec{x} - \vec{p}) \cdot \vec{n} = 0$$



**Figure 46:** Example setting for *Recursive Light Ray* algorithm.

where  $\vec{p}$  is a point on the plane  $E$  and  $\vec{n}$  is the normal vector of  $E$ . The  $PBP(x_2, x_1)$  is given by

$$\vec{p} = \vec{x}_2 + \frac{1}{2}(\vec{x}_1 - \vec{x}_2).$$

and

$$\vec{n} = \vec{x}_1 - \vec{x}_2.$$

So,

$$PBP(x_2, x_1) : (\vec{x} - \vec{x}_2 + \frac{1}{2}(\vec{x}_1 - \vec{x}_2)) \cdot (\vec{x}_1 - \vec{x}_2).$$

So, we compute the equations of all  $PBP(x_2, \cdot)$ . For the example, we need the results for  $PBP(x_2, x_0)$  and  $PBP(x_2, x_1)$ .

$$PBP(x_2, x_0) : \left( \vec{x} - \begin{pmatrix} 1/2 \\ 1/2 \end{pmatrix} \right) \cdot \begin{pmatrix} -1 \\ -1 \end{pmatrix}$$

$$PBP(x_2, x_1) : \left( \vec{x} - \begin{pmatrix} 1/2 \\ 1 \end{pmatrix} \right) \cdot \begin{pmatrix} -1 \\ 0 \end{pmatrix}$$

Now, we have to define the initial set of *Light Rays*. With the help of this vectors, we want to detect each corner of the current Voronoi cell. The initial set of *Light Rays* is given by unit vectors with positive and negative sign. In our example, we get:

$$\begin{aligned}\vec{L}_1 &= \vec{x}_2 + r \cdot \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \end{pmatrix} + r \cdot \begin{pmatrix} 1 \\ 0 \end{pmatrix} \\ \vec{L}_2 &= \begin{pmatrix} 1 \\ 1 \end{pmatrix} + r \cdot \begin{pmatrix} 0 \\ 1 \end{pmatrix} \\ \vec{L}_3 &= \begin{pmatrix} 1 \\ 1 \end{pmatrix} + r \cdot \begin{pmatrix} -1 \\ 0 \end{pmatrix} \\ \vec{L}_4 &= \begin{pmatrix} 1 \\ 1 \end{pmatrix} + r \cdot \begin{pmatrix} 0 \\ -1 \end{pmatrix}\end{aligned}$$

Now, we are able to derive the first intersection points of the *Light Rays* and the *PBPs*. In general, the algorithm has to compute all intersections between each *Light Rays* and each  $PBP(x_2, .)$ . We want to find the *PBP* for each *Light Ray* with the smallest but positive parameter value for  $r$  of the corresponding *Light Ray*. In our example we compute the intersection point of  $\vec{L}_3$  with  $PBP(x_2, x_0)$  and  $\vec{L}_3$  with  $PBP(x_2, x_1)$ .

$$\begin{aligned}PBP(x_2, x_0) : \left[ \vec{x} - \begin{pmatrix} 1/2 \\ 1/2 \end{pmatrix} \right] \cdot \begin{pmatrix} -1 \\ -1 \end{pmatrix} \\ PBP(x_2, x_0) : x + y = 1\end{aligned}\tag{97}$$

where  $\vec{x} = (x, y)^T$ .

The intersection point between  $\vec{L}_3$  with  $PBP(x_2, x_0)$   $s_{2,3} = (0, 1)^T$ . The intersection point is named after the index of the light ray and the corresponding *PBP*. The distance between  $x_2$  and the intersection point  $s_{2,3}$  is given by  $|x_2 - s_{2,3}| = 1$ . The parameter  $r$  of *Light Ray*  $\vec{L}_3$  is 1. In the next calculation  $\vec{L}_3$  intersects with  $PBP(x_2, x_1)$  at  $s_{1,3} = (1/2, 1)^T$ . The distance is  $|x_2 - s_{1,3}| = 1/2$ . The parameter  $r$  is smaller with  $1/2$ . Therefore, the algorithm stores  $PBP(x_2, x_1)$  as shortest *PBP* for light ray  $\vec{L}_3$ . The computation of all other intersection points for  $\vec{L}_3$  finds no smaller and positive value for  $r$ .

If we continue these calculations for all *Light Rays* and all  $PBP(x_2, .)$ , we obtain a list of closest intersection points of each *Light Ray* as follows:

$$\begin{aligned}\vec{L}_1 \cap PBP(x_2, x_4) &= s_{4,1} = \begin{pmatrix} 3/2 \\ 1 \end{pmatrix} \\ \vec{L}_2 \cap PBP(x_2, x_3) &= s_{3,2} = \begin{pmatrix} 1 \\ 3/2 \end{pmatrix} \\ \vec{L}_3 \cap PBP(x_2, x_1) &= s_{1,3} = \begin{pmatrix} 1/2 \\ 1 \end{pmatrix} \\ \vec{L}_4 \cap PBP(x_2, x_0) &= s_{0,4} = \begin{pmatrix} 0 \\ 1 \end{pmatrix}\end{aligned}$$

This is the end of the 0.iteration step. We obtain a list of probable nearest neighbors, but without any kind of verification. Therefore, we continue with the verification of the probable

nearest neighbors. The probable vertices of the corresponding Voronoi cell of  $\vec{x}_2$  are given by the intersection points of each *PBP* of each nearest neighbor. If a *Light Ray* does not intersect with a *PBP* with a positive value for  $r$ , the algorithm has detected an open Voronoi cell. These kind of cells can be found at the border region of the data space. In order to close these kind of cells, we have to introduce *Bounding Planes*  $BP : x_i = -MAX$  or  $BP : x_i = +MAX \forall i \in [0, d)$ . These *BPs* limit the data space at a defined distance  $MAX$ . Since all *BPs* are given by one coordinate, all *BPs* are orthogonal to each other. The algorithm computes all intersection points with the *Light Ray* and each bounding plane *BP*. The *BP* with the shortest but positive distance is selected. Now we continue with the computation of the probable vertices of our Voronoi cell by intersecting the *PBP*, which have been detected by the *Light Rays*. In our short example, we are using the detected *PBPs* of *Light Ray*  $\vec{L}_3$  and  $\vec{L}_4$ . So, we intersect  $PBP(x_2, 1)$  and  $PBP(x_2, x_0)$ . In case of the detection of Voronoi cells in higher dimension, the algorithm has to intersect as many *PBPs* as dimensions of the data space. In our example the *PBPs* are given by:

$$\begin{aligned} PBP(x_2, x_1) : x_1 &= 1/2 \\ PBP(x_2, x_0) : x_1 + x_2 &= 1 \end{aligned}$$

Consequently, both planes intersect at

$$S_{2,0,1} = \begin{pmatrix} 1/2 \\ 1/2 \end{pmatrix}$$

where  $S_{2,0,1}$  defines the intersection point of the  $PBP(x_2, x_0)$  and  $PBP(x_2, x_1)$ . Now, at the beginning of the first iteration step, we define a new *Light Ray*, which goes from  $\vec{x}_2$  to  $S_{2,0,1}$ :

$$\vec{L}_{2,0,1} = \begin{pmatrix} 1 \\ 1 \end{pmatrix} + r \cdot \begin{pmatrix} -1/2 \\ -1/2 \end{pmatrix} \quad (98)$$

Before, we continue, the algorithm has to verify, that the intersection point of the *PBPs* can be written as linear combination of the *Light Rays*, which have been used to detect the corresponding planes with positive multiplication factors. In detail, we have to check if the parameters  $a$  and  $b$  in

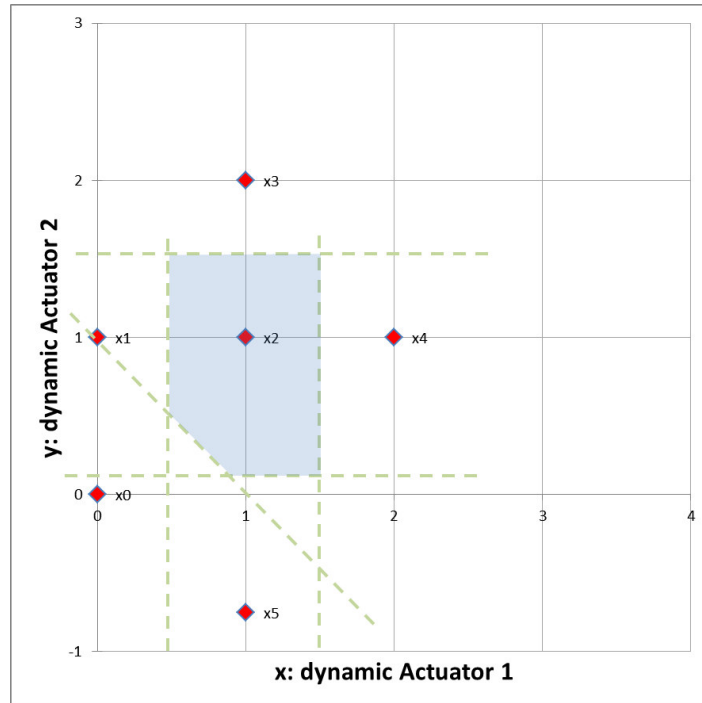
$$a \cdot \vec{L}_3 + b \cdot \vec{L}_4 = S_{2,0,1}$$

are positive. Otherwise, the detected *PBPs* are divergent, which means, that the intersection point is not in the direction of the *Light Rays*. In this case, we cannot use the *Light Ray*  $\vec{L}_{2,0,1}$ . We have to derive a random light ray, which is given by a positive linear combination of the corresponding light rays  $\vec{L}_3$  and  $\vec{L}_4$ . If this random *Light Ray* detects a new closest *PBP* we can continue with the computation of the new intersection points  $S'_1$  and  $S'_2$ . These new intersection point replace the originally detected intersection point  $S_{2,0,1}$ . If the random *Light Ray* does not detect a new plane, the algorithm has to determine a new random light ray and repeat the process until a new *PBP* has been detected. All detected intersection points are passed to the next function, which validate possible vertices of the Voronoi cell. With this new *Light Ray* / *Light Rays* we try to verify the possible corner at  $S_{2,0,1} / S'_1$  and  $S'_2$ . Our assumption is: If  $\vec{L}_{2,0,1}$  points to a corner of the Voronoi cell of data point  $\vec{x}_2$ , it will not intersect with any other *PBP* with a smaller but positive value of parameter  $r$  of the new *Light Ray*. Furthermore, the new *Light Ray* will intersect as many *PBPs* in one point  $p$  as



dimensions of the data space. Due to the limited calculation accuracy of computer systems, we define a small neighborhood  $\epsilon$  around  $p$ . If all necessary *PBP* share their intersection point with the new *Light Ray* in this small environment, the corner is verified. All necessary variables of this computation are given as *Double* variable type. In our example, we are intersecting the new *Light Ray*  $\vec{L}_{2,0,1}$  with  $PBP(x_2, x_1)$  and with  $PBP(x_2, x_0)$ .  $\vec{L}_{2,0,1}$  intersects  $PBP(x_2, x_1)$  at  $(1/2, 1/2)^T$  and  $PBP(x_2, x_0)$  at  $(1/2, 1/2)^T$ . So, both intersection points are equal and the computation of all other intersection points with  $\vec{L}_{2,0,1}$  and  $PBP(x_2, \cdot)$  show, that no smaller but positive parameter values for  $r$  exists. So, we may conclude, that  $\vec{x}_2$  and  $\vec{x}_1$  as well as  $\vec{x}_2$  and  $\vec{x}_0$  are nearest neighbors. This calculation has to be continued for remaining set of *Light Rays* in order to detect all nearest neighbors of  $\vec{x}_2$ .

Of course, we cannot hope to find all vertices of the corresponding Voronoi cell in the first iteration step. In order to make this point clear, we introduce a new example setting. In this



**Figure 47:** Advanced example setting for *Recursive Light Ray* algorithm.

new example setting (see Figure 47), we add an additional  $PBP(x_0, x_5)$ , which cuts a small region of our former Voronoi cell. As before, we want to compute the nearest neighbors of  $\vec{x}_2$ .

In the 0.iteration step the unit vectors or *Light Rays*  $\vec{L}_4$  and  $\vec{L}_3$  detect  $PBP(x_2, x_5)$  and  $PBP(x_2, x_1)$  as closest *PBP* to  $\vec{x}_2$ . However, the intersection point of  $PBP(x_2, x_5)$  and  $PBP(x_2, x_1)$  is not a corner of the Voronoi cell of  $\vec{x}_2$ .

As regular, the algorithm defines a new *Light Ray*, which has to verify the corner, but in this case, this light ray detects  $PBP(x_2, x_0)$  with a smaller parameter value  $r$ . This is the starting point for the next iteration step. The algorithm defines additional *Light Rays*, which point from  $\vec{x}_2$  to the intersection point of  $PBP(x_2, x_1)$  and  $PBP(x_2, x_0)$  as well to the intersection point of  $PBP(x_2, x_0)$  and  $PBP(x_2, x_5)$ . The maximal iteration depth is configured to be 50 steps.

---

After the algorithm has computed all *Light Rays* of each data point, it delivers the complete set of nearest neighbors of the data points.

**Assessment of the Recursive Light Ray Method:** This algorithm has been introduced by this work in order to determine the nearest neighbors of a given data set. The main goal of this algorithm is the fast detection of nearest neighbors for high dimensional data sets. Therefore, we introduce a method, which can be computed in parallel. So, we are able to compute each Voronoi cell and each *Light Ray* in parallel on *Graphical Processing Units GPUs*. Furthermore, we are using a lot of simple computation steps, e.g. intersections between lines and planes, but less more complex computation steps, e.g. intersection computation of planes. That is why the algorithm is able to compete with established State-of-the-Art methods. This aspect will be studied in Benchmark subsection (see Subsection 5.2.1). On the other hand, we did not include a mechanism, which handle the open Voronoi cells at the border of the data space. So, it is possible, that the algorithm does not detect all nearest neighbors at the border region of the data space. This aspect will be investigated in Subsection 5.1.7. This point is not crucial for this work, since the interpolation processes, which cause the Voronoi tessellations are forbidden in the border regions of the data space, since the probability of extrapolations is very high at the border regions. On the other hand, we established a method, which does not predict wrong nearest neighbors. Because the algorithm has to find each corresponding corner of the Voronoi, This is very important, in order to avoid wrong interpolation processes.

#### 4.4.1.2 Description of Recursive Light Ray Program

##### 1. Basic Idea of Algorithm

In this section, we introduce the key points of our implementation of the *Recursive Light Ray* algorithm. This algorithm has been explained in the previous passage.

Basically, this algorithm derives the vertices of each Voronoi cell according to a data point  $d_i$  (see Figure 48). Therefore, the algorithm determines the perpendicular bisector planes of each point  $d_k$  to  $d_i$   $PBP(d_k, d_i)$ . Afterwards, the algorithm has to determine the closest  $PBP$  to  $d_i$ . Finally, the corners, which are given by the intersection points of the corresponding perpendicular bisector planes, have to be verified. A corner is valid, if a vector exists from  $d_i$  to the corner without intersections of other planes between  $d_i$  and the corner. The whole algorithm and its different modi are explained in detail in the following.

##### 2. Database of Algorithm

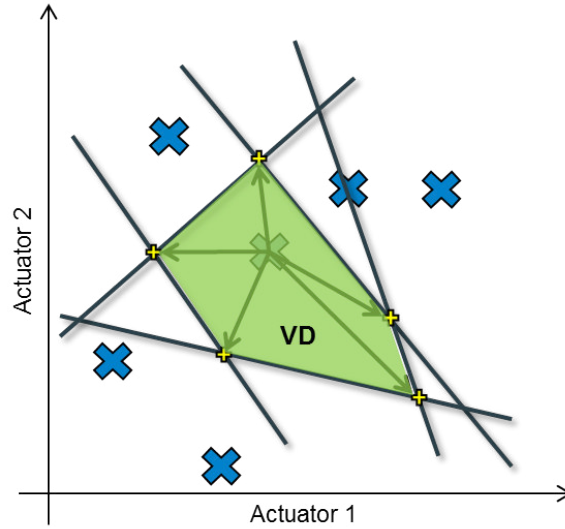
The algorithm works on discrete data that has been filtered by time series compressor (see Subsection 4.3.1) and the adaptive space compressor (see Subsection 4.3.2). All data points are rounded to a suitable amount of decimal digits, which are given by the physical and technical specifications.

##### 3. Implementation Language and Parallelization

The algorithm has been implemented in *Python* and *OpenCL*. The implementation is able to calculate all vertices of a *Voronoi* cell in parallel.

##### 4. Modi of Algorithm

The implementation consists of two different calculation modi: the initial determination and the iterative extension of the Voronoi graph. The initial determination mode derives the nearest neighbors of a set of data points. Additionally, it takes the distance



**Figure 48:** Drawing of a Voronoi cell in order to illustrate the method of the algorithm

of the data points into account. Nearest neighbor determination is skipped, if the distance between the data points is too large. The second mode, the iterative extension of an existing graph, works in the same way than the first mode. Yet, it is able to expand an existing graph. Both modi are explained in the following paragraphs.

## 5. Initial Determination of nearest Neighbors

In order to illustrate the implementation of the algorithm, we explain the method of the determination of all vertices of one example data point  $d_i$ . The whole algorithm is divided into a *Python* and an *OpenCL* part. Detailed flowcharts of the algorithms are illustrated in the Appendix of this work. *Python* organizes the data, but the determination process of the *Voronoi* cells is evaluated by *OpenCL* functions. In general, *Python* transmits the measured data points *data* and an empty matrix *Neighbors* to the GPU memory. So, these data structures are accessible by the *OpenCL* functions. Afterwards, *OpenCL* retransmits the complete nearest neighbor information *Neighbors\_buf* back to *Python*. The most important *OpenCL* functions will be introduced in the following.

First, the algorithm calls the function *create\_rays*.

### a) *create\_rays*

This function creates a basic set of vectors, which are called *light rays*  $L_j$  during the further procedure. The dimension  $d$  of the data space is given by the number of *dynamic* Actuators of the optimization process. This function produces  $2 \cdot d$  vectors, which are given by unit vectors of each dimension with positive and negative sign. An example for  $n = 2$  dimensional data points is given in the following:

$$L_i = [\vec{e}_1, \vec{e}_2, -\vec{e}_1, -\vec{e}_2] = \left[ \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \end{pmatrix}, \begin{pmatrix} -1 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ -1 \end{pmatrix} \right] \quad (99)$$

Furthermore, these *light rays* are used to determine the closest perpendicular bisector planes from data point  $d_i$  to each other data points. Therefore, we have to calculate the intersection points of each *light ray* with each perpendicular bisector plane. This is computed in the following function *create\_intersections*.

b) *create\_intersections*

The intersection of a vector *light ray*  $\vec{L}_j$  with the perpendicular bisector planes has to be derived. The general equation of a line is given by:

$$\vec{x} = \vec{d}_i + s\vec{L}_j \quad s \in \mathbb{R} \quad (100)$$

If we insert Equation (100) into the equation of the perpendicular bisector plane, we obtain

$$\frac{1}{2}(d_i^T d_i - d_k^T d_k) = (d_i - d_k)^T (d_i + sL_j) \quad (101)$$

$$= d_i^T d_i - d_k^T d_k + d_i s L_j - d_k s L_j. \quad (102)$$

So, we are able to determine the intersection point  $s$  of a *light ray*  $\vec{L}_j$  and our perpendicular bisector plane.

$$\Rightarrow s = \frac{\overbrace{\frac{1}{2}d_i^T d_i}^{ptp1} + \overbrace{\frac{1}{2}d_i^T d_i - d_k^T d_k}^{ptp2}}{\underbrace{(d_i - d_k)^T L_j}_{ptp3}} \quad (103)$$

In the algorithm, we compute the parameter  $ptp1$ ,  $ptp2$  and  $ptp3$  in order to determine the intersection point  $s$ .

- i. Now, we have to determine the perpendicular bisectors of point  $d_i$  to each other data point and afterwards, we have to calculate the intersection point  $s$  of each *light ray* to the perpendicular bisectors.

A. The data point  $d_i$  is given by its components  $d_i = (x_i^1, \dots, x_i^n)^T$ . The data point  $d_j$  of the considered perpendicular bisector plane is given by  $d_j = (x_j^1, \dots, x_j^n)^T$ . Consequently, the first parameter  $ptp1$  is given by

$$ptp1 = \sum_{j=0}^{dim} \frac{1}{2} \cdot (x_i^j)^2.$$

- B. In order to determine the parameters  $ptp2$  and  $ptp3$  we have to loop over all *light rays* with the index  $j$  and within this loop, we have to loop over all other data points with the index  $k$ . So, we are able to determine

$$ptp2 = \sum_{h=0}^{dim} \left( \frac{1}{2} \cdot x_k^h - x_i^h \right) \cdot x_k^h \quad (104)$$

and

$$ptp3 = \sum_{h=0}^{dim} (x_k^h - x_i^h) \cdot L_j^h. \quad (105)$$

Furthermore, we determine the current euclidean distance between  $d_i$  and  $d_k$ :

$$distance = \sum_{h=0}^{dim} (x_k^h - x_i^h)^2. \quad (106)$$

If *distance* is greater than a user specified maximal distance  $d_{max}$  or the parameter  $ptp3 = 0$ , we interrupt the evaluation of the current perpendicular bisector plane and continue with the next data point  $d_k$ . Otherwise, we are able to evaluate the intersection point  $s$ . The distance between  $d_i$  and the intersection point  $s$  is given by:

$$ptp4 = \frac{ptp1 + ptp2}{ptp3}. \quad (107)$$

- ii. For each *light ray* we store the index of the closest perpendicular bisector planes. So, we have to remember data point  $d_k$  with the smallest but positive value of the parameter  $ptp4$ . In the case of more than one point that meets the requirement within the calculation accuracy  $\epsilon$ , both values are stored. If more than one values is stored, these data points form a possible corner of the Voronoi cell.
- iii. This routine *create\_intersections* returns a list of the closest perpendicular bisector planes detected with one *light ray* around data point  $d_i$ . These planes are candidates for the determination of possible vertices of the Voronoi cell. In the following steps of the algorithm, we have to combine the detected planes, in order to compute the intersection points of the detected perpendicular bisector planes. These plane combinations are determined and tested in the next routines *calc\_corners* and *determine\_corners*.

c) *calc\_corners / determine\_corners*

For the next step of the evaluation, we have to determine the possible vertices of the current Voronoi cell. Therefore, we test a subset of the selected closest perpendicular bisector planes from the *create\_intersections* function. We need  $d$  planes in a  $d$ -dimensional space to determine the intersection point of the planes. These intersection points represent the possible limits of the Voronoi cell. The following function *binary* does the selection of the planes, which have to be combined.

i. *binary*

This function determines the combination of perpendicular bisector planes for the evaluation of the possible vertices of the Voronoi cell. Therefore, this function goes through all possible combinations of *light rays*. Basically, we have a sequence of integer numbers, which are starting at 0 and end until all combinations have been evaluated. In this loop, we translate each integer number into a binary number. The first digit defines the sign of the first *light ray*, 0 means negative, 1 means positive. The next digits define the signs of the following *light rays*. Each *light ray* has intersected with at least one closest perpendicular bisector plane, therefore we have to select as many *light rays* as dimensions of the data space. In the previous function, we store the closest planes, in accordance to the applied *light ray*. An example for  $n = 2$  dimensional data points is given in following table:

**Table 6:** Example for binary function ( $n = 2$ )

binary	light rays / unit vectors
00	$-\vec{e}_1$ and $-\vec{e}_2$
01	$-\vec{e}_1$ and $+\vec{e}_2$
10	$+\vec{e}_1$ and $-\vec{e}_2$
11	$+\vec{e}_1$ and $+\vec{e}_2$

In the next steps of the algorithm, we have to determine the plane equation of the selected plane combinations. So, we are able to determine the intersection points of the given plane combinations. The plane equation is derived by the following function *determine\_plane\_equation*.

d) *determine\_plane\_equation*

This function prepares the linear equation system for each defined combination of the *binary* function. These equation systems have to be solved in order to determine the intersection point of the selected planes.

- i. The linear equation system is given by the selected planes from the previous function. The homogeneous part of the equation system is given by

$$hom\_part_k^m = x_i^m - x_{P_s^k}^m, \quad (108)$$

where  $P_s^k$  is the set of the indices of the closest perpendicular bisector planes of *light ray* with index  $k$ . The in-homogeneous part of the equation system is given by

$$inhom\_part_k = \sum_{k=0}^{dim} \sum_{m=0}^{dim} \frac{1}{2} \left[ (x_i^m)^2 - (x_{P_s^k}^m)^2 \right]. \quad (109)$$

The complete equation system is solved by the function *Gauss\_solver*.

ii. *Gauss\_solver*

The defined linear equation systems of the *determine\_plane\_equation* function are solved with the Gaussian elimination process. The solved equation systems define possible vertices  $sp$  of the selected Voronoi cell of data point  $d_i$ . As introduced in the beginning of this passage, we have to proof each corner of the Voronoi cell by a direct *light ray* from data point  $d_i$  to the corner. This is done in the following functions.

iii. *calc\_intersections*

The determined intersection points of the perpendicular bisector planes with the *light rays*  $sp$  have to be proven as vertices of the Voronoi cell. Therefore, we compare the set of closest planes  $P_s^k$  of *light ray*  $k$  with the indices of the perpendicular bisector planes, which contribute to the corresponding possible corner  $sp$ . As introduced in the beginning of this passage, the algorithm has found a corner of the Voronoi cell, if a *light ray* exists from the data point  $d_i$  and the detected corner  $sp$ , without intersecting any other perpendicular bisector plane between  $d_i$  and  $sp$ . That is why, we define a new *light ray* from  $d_i$  to  $sp$ . If this light ray intersects with more than one perpendicular bisector

plane at the same location, within an  $\epsilon$  environment, we conclude, that this could be a corner of the cell. Furthermore, if the same planes intersect at the same location  $sp$ , we are able to proof the corner as corner of the Voronoi cell. If this new *light ray* intersects another plane  $sp_k$  at a shorter distance than  $sp$ , we have to conclude, that  $sp$  is not a corner of the Voronoi cell. Since, this corner has been falsified, the algorithm has to continue with the next iteration step. The iteration step starts with the creation of new *light rays*. In contrast to the first iteration step, the *light rays* are no longer given by unit vectors. The new *light rays* are given by the intersection points of the planes  $sp$  and  $sp_k$ . So, we produce  $d$  new *light rays* that point to the new possible vertices of the Voronoi cell. The validation or falsification of the new possible vertices are done by the by the functions *calc\_intersections*, *calc\_corners*, and *determine\_corners*. The iteration process continues until all vertices of the Voronoi cell have been detected or the maximal iteration limit has been reached.

e) **Iterative Extension of existing Vornoi Graph** The iterative extension mode of the algorithm expands an existing Voronoi graph. So, additional data points can be added. This mode of the algorithm uses basically only routines of the initial determination mode.

i. *Damage\_grid*

First, the nearest neighbors of the new data points are derived by evaluation of the functions, which are described in the points 5a to point 5(d)ii. The new function *damage\_grid* loops over all new data points, but the index starts with the last index of the old data set. Hence, the algorithm determines all nearest neighbors of the new data points among each other and moreover it determines all nearest neighbors of the new data points to the old data set *neighbors\_to\_erase*. This process is identical to the initial mode of the *Recursive Light Ray* algorithm. It must be pointed out that the nearest neighbors of the old data points among each other are incorrect now, if the old data point is the nearest neighbor of any new data point. This defect has to be corrected in the next algorithm processes.

ii. *Erase\_partial\_old\_net*

Now, the algorithm focuses on the old data points that are the nearest neighbors of any new data point. The determined nearest neighbor information of these points is erased. The deletion process is performed by creating a triangle matrix *big\_ones* that is given by

$$big\_ones_i^j = \begin{cases} 1 & \text{if } d_i \text{ nearest neighbor of } d_j \text{ and } d_i \text{ is old data point} \\ 1 & \text{if } d_j \text{ nearest neighbor of } d_i \text{ and } d_j \text{ is old data point} \\ 0 & \text{else.} \end{cases}$$

This matrix *big\_ones* contains information, if a pair of two data points  $d_i$  and  $d_j$  are nearest neighbors under the restriction, that either  $d_i$  or  $d_j$  is a new data point and the other data point is an old data point.

In each inverse entry of *big\_ones* and in each entry of the existing neighbor matrix *neighbors*, we apply the logical *and* operator:

$$damaged\_grid_i^j = \text{not}(big\_ones_i^j) \text{ and } neighbors_i^j \quad (110)$$



So, we derive the new incomplete Voronoi graph *damaged\_grid*, which does not include any wrong connection. In the final step of the iterative mode, the algorithm has to repair the incomplete Voronoi graph *damaged\_grid*. This is done by the following function *Repair\_grid*.

iii. *Repair\_grid*

Finally, the nearest neighbors of all erased old data points are recalculated, by using the routines, which have been introduced in the points 5a until point 5(d)ii with the incomplete Voronoi graph *damaged\_grid*. The routine *Repair\_grid* loops over the erased old data points. The resulting matrix *damaged\_grid* is combined with the matrix *neighbors\_to\_erase* by applying the logical *or* operator on each entry of the matrix:

$$\text{Repaired\_grid}_i^j = \text{Damaged\_grid}_i^j \text{ or } \text{neighbors\_to\_erase}_i \quad (111)$$

The matrix *Repaired\_grid* contains the whole nearest neighbor information of the old and new data set. The result is equivalent to a determination of the complete data set with the initial mode of the *Recursive Light Ray* algorithm. This mode is studied in detail in the following result chapter. The validation of the algorithm is done in Subsection 5.1.7. Benchmark calculations are done in 5.2.1.

---

#### 4.4.2 Determination of Tolerance Table

---

In order to conserve the maximal gradient constraint (see Equation (8)), we have to compute all actuator  $a_m \in A$  gradients  $\nabla_{ij}^m$  with respect to all solution surface variables  $s_k \in S$ . The calculation of the current gradients between all points is split into several steps.

The algorithm loops over all valid data points  $d_i \in D$ . A second loop goes over all data points  $d_j \in D$  but not for  $i = j$ . First, the algorithm determines the euclidean distance  $\text{dist}_s$  relating to the solution surface between data point  $d_i$  and  $d_j$ :

$$\text{dist}_s(d_i, d_j) = \sqrt{\sum_{k \in S} (d_i^k - d_j^k)^2} \quad (112)$$

Afterwards the algorithm determines the deviation  $\delta_{ij}^m$  between the settings of the corresponding actuator  $m$  of point  $d_i$  and  $d_j$ :

$$\delta_{ij}^m = |d_i^m - d_j^m|. \quad (113)$$

Finally, each deviation  $\delta_{ij}^m$  is divided by the euclidean distance  $\text{dist}_s(d_i, d_j)$ , the ratio is compared to the user definition of the maximal gradient  $\nabla_{max}^m$  of the actuator  $m$ .

$$\nabla_{ij}^m = \frac{\delta_{ij}^m}{\text{dist}_s(d_i, d_j)} \quad (114)$$

If any component gradient is higher than the user defined maximal gradient, the connection between data point  $d_i$  and  $d_j$  is defined as invalid. The information is stored as matrix  $T_{ij} \in \mathbb{B}^{n \times n}$ :

$$T_{ij} = \{t_{ij} | \nabla_{ij}^m \leq \nabla_{max}^m \forall i, j \in n\} \quad (115)$$

where  $n$  is the number of data points. Due to the high computational effort, this algorithm has been implemented in *OpenCL*, therefore this algorithm runs on GPU with a high degree of parallelism. The resulting triangle matrix is stored in an indexed array structure.



---

#### 4.4.3 Definition of Linear Integer Optimization Problem

---

This subsection is based on the definition of the mathematical problem in Subsection 1.1. In this passage, we specify the description of the mathematical problem as integer program IP. The optimization of an engine map can be written as linear integer optimization problem. A linear integer optimization problem consists of an objective function, linear constraints and variable definitions. The defined integer optimization problem, will be described in the following paragraph. Afterwards, we solve the IP with the *Branch and Cut* method.

The measured data points are given by vectors  $d$  in  $\mathbb{R}^n$ . Each vector  $d$  contains dynamic actuator settings  $A_d^j$ , static actuator settings  $A_s^k$  and measured sensor values  $M^l$ .

$$d = (A_d^0, A_d^1, \dots, A_d^{j-1}, A_s^0, A_s^1, \dots, A_s^{k-1}), \quad (116)$$

$$M^0, M^1, \dots, M^{l-1} \in \mathbb{R}^{j+k+l}$$

$D$  is defined as the set of all vectors  $d$ . All data points  $d \in D$  are grouped with respect to their location in the solution. A stack or operational point  $s_{kl} \in S$  defines a rectangle region in the solution map.  $k$  defines a certain interval of revolution frequencies  $\nu$ ,  $l$  defines an interval of engine torque  $M$ . So a stack  $s_{kl}$  can be defines as:

$$s_{kl} := \{x \in \mathbb{R}^n | x \in [b_k^\nu, b_{k+1}^\nu] \text{ and } x \in [b_l^M, b_{l+1}^M]\} \quad (117)$$

Data points  $d$  are grouped in a stack  $s_{kl}$  if its revolution frequency and its engine torque is within the given intervals of the stack  $s_{kl}$ .  $S$  is the set of all stacks  $s_{kl} \in S$ .

The solution map is given by stacks of valid data points where  $N_k$  is the set of valid data points  $d_{ik}$  in stack  $s_k$ . The decision of the optimization algorithm is modeled with binary variables  $x_i \in \{0, 1\}$ .  $x_i = 1$  means, that the data point will be part of the solution otherwise, the data point will be ignored. The position and subset of data points is defined by the configuration of the solution field. Data points of one region is unified to one representative stack of this region. In order to choose only one data point per region, the optimization process has to fulfill the *stack constraint*

$$X_k + \sum_{i \in N_k} x_i = 1, \forall k \in S \quad (118)$$

where  $X_k$  is a binary variable that has to be 1 if it is not possible to choose exactly one data point for stack  $s_k$ . In the following, we call this variable *penalty stack point*. Equation (118) defines, that up to one data point  $x_i$  may be chosen from stack  $s_k$ . If no data point has been chosen  $X_k$  has to be 1. Due to bad data space coverage or other constraints, it is possible, that no valid data point can be chosen from a stack, so the *penalty stack point* has to be chosen. This point has no prey value (objective function) but defined high emission values.

##### 4.4.3.1 Objective Function

In case of integer problems, the solver chooses data points that maximize or minimize the criterion of the objective function. In our specific case case, we want to minimize the specific fuel consumption in each sub region of the solution. Therefore, we apply the definition of stacks  $s_k$  (see Equation (118)) and the definition of the *prey* value  $p_i$  (see Equation (9)). The objective function  $z$  is given by:

$$z = \sum_{i=0}^{|D|} p_i x_i \rightarrow \max, \text{ with } x_i \in \{0, 1\} \quad (119)$$

#### 4.4.3.2 Constraints

The optimization problem has been introduced with the following side conditions: integral emission condition, critical data point condition, maximal gradient condition and stack constraint (see Subsection 1.1). These constraints have to be fulfilled in order to acquire a feasible solution.

The integral emission condition describes the upper bound of several emission test cycles, e.g. maximal  $NO_x$  production. This emission time integral value is given by the official test cycles, e.g. New European Driving Cycle NEDC or Real World Driving Cycle (see 2.2.3). In Subsection 2.2.3.1 we introduced the calculation of the weighting factors  $u_i^j$ , which represent the mean resistance time of the operation points  $O(v, M)$  in the given test cycle. The monitored emission species  $j$  in the set of chemical species  $E$  of each data point  $d_i$  are stored in  $d_i^j$ . The upper bound of the integral exhaust emission value of emission species  $j$  is given by  $\bar{d}_j$ . Formally, the integral emission constraint can be written as:

$$\sum_{i=0}^{|D|} u_i^j d_i^j x_i + \sum_{s_k \in S} u_{s_k}^j \bar{d}_j X_k \leq \bar{d}_j^{int}, \forall j \in E \quad (120)$$

where  $\bar{d}_j^{int}$  is the upper threshold of integral exhaust emission value of emission species  $j$ .

One more constraint is the critical data point condition, which eliminates data points that violate critical upper or lower boundary limits of sensor values  $\bar{p}_j$ . This constraint can be written as:

$$d_i^j x_i \leq \bar{p}_j, \forall j \in |d| \quad (121)$$

In order to maintain driveability, the chosen actuator configuration has to fulfill the maximal gradient constraint. This constraint is based on the tolerance table  $T_{ij}$  (see Equation (114) and Equation (115)). The solution engine map has to fulfill physical reaction times of the engine. In general, actuators may be varied much faster than the physical system is able to react to the variation of the actuators. This constraint equals the maximal variation speed with the physical reaction time of the engine.

$$x_i + x_j \begin{cases} \leq 1 & \text{if } T_{ij} = 0 \\ \leq 2 & \text{if } T_{ij} = 1 \end{cases} \quad (122)$$

An example for a whole model will be presented in the following.

```

Maximize
1.0  $x_0$  + 0.9980  $x_1$  + ... + 0.9964  $x_{1017}$ 
Subject To
E0: 1.0  $X_0$  + 1.0  $X_1$  + 1.0  $X_2$  + ... + 1.0  $X_{210}$  + 1.0  $X_{211}$  + 0.004480  $x_{35}$  + 0.002228  $x_{40}$ 
+ ... + 0.002281  $x_{975}$  + 0.003666  $x_{977} \leq 1000.0$ 
...
E3: ...
S0:  $X_0 + x_{201} + x_{202} + x_{203} + x_{204} + x_{205} = 1$ 
...
S211: ...
T17_329:  $x_{1017} + x_{329} \leq 1$ 
...
T47_45:  $x_{47} + x_{45} \leq 1$ 
binary
 $X_0 X_1 \dots X_{210} X_{211} x_0 x_1 \dots x_{1016} x_{1017}$ 
End

```

The first line shows, that the linear integer problem has to be maximized. The numerical value is given by  $p_i$  of Equation (9). In the next lines, one defines the side conditions of the optimization problem. The side constraints  $E..$  handle maximal integral value of certain emission species (see. Equation (120)). Summands with a small  $x$  represent measured data points, Summands with a capital  $X$  represent *penalty stack decisions* (show Equation (118)). The following side constraints are given by the stack formation. These constraints start with  $S$ . These constraint guarantee that one data point in a stack is chosen, exactly. In case of empty stacks or other constraint violations, the solver is forced to take the *penalty stack decision*. The final pack of constraints are defined by the driveability constraint (see Equation (122)). These constraints start with  $T$ . These side conditions exclude data points, which would violate the maximal gradient condition among all data points. The final paragraph define all variables as binary variables.

---

## 5 Results

---

### 5.1 Validation of Methods

---

The proposed methods in Section 4 have to be validated in order to guarantee functionality. Typically, a method has a specific purpose and a well-defined environment. We define the environment and an assumption, which has to be fulfilled by the method.

#### 5.1.1 Dynamic Test Drive

---

The general purpose of *Dynamic Test Drive* is the coverage of the data space via measurement ramps. Therefore, this method creates a data histogram of the data space and creates weighted random points, which define start-/end points of the measurement ramps. A more detailed discussion is given in Subsection 4.2.

##### 5.1.1.1 Environment

This method analyses data points of a  $j$ –dimensional data space with  $j$  dynamic actuators,  $k$  static actuators and  $l$  result observables. The specification of the data histogram is taken from the asymmetrically split data tree *Tree*. The histogram is given by the leaves of *Tree* (see Subsection 4.2.1). All data points are sorted into the data tree.

##### 5.1.1.2 Assumption

The algorithm *Dynamic Test Drive* has to create measurement ramps that cover the data space as homogeneous as possible. A homogeneous data coverage is the basis of further optimization steps.

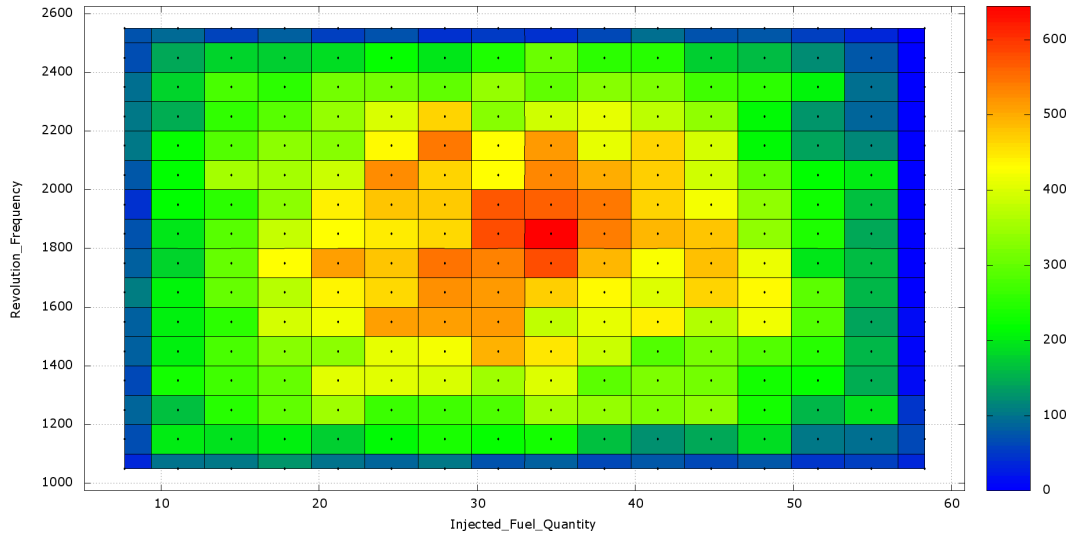
##### 5.1.1.3 Proof of Principle

In order to proof this measurement technique, we show the impact of weighted random numbers, by comparing the data density of two measurement runs. Both measurement runs took 96 hours and two dynamic actuators have been varied, i.e. *Revolution Frequency* and *Injected Fuel Quantity*. The first run is based on uniformly distributed random numbers, the second one is based on weighted random numbers. The weighting function is given by the inverse data density (see Subsection 4.2.2).

Figure 49 shows the data density of the measurement run without weighted random numbers. The creation of measurement ramps is explained in 4.2.2. Figure 50 shows the data density of the measurement run with weighted random numbers. The data space resolution of both runs has been equally chosen with 16 bins per actuator.

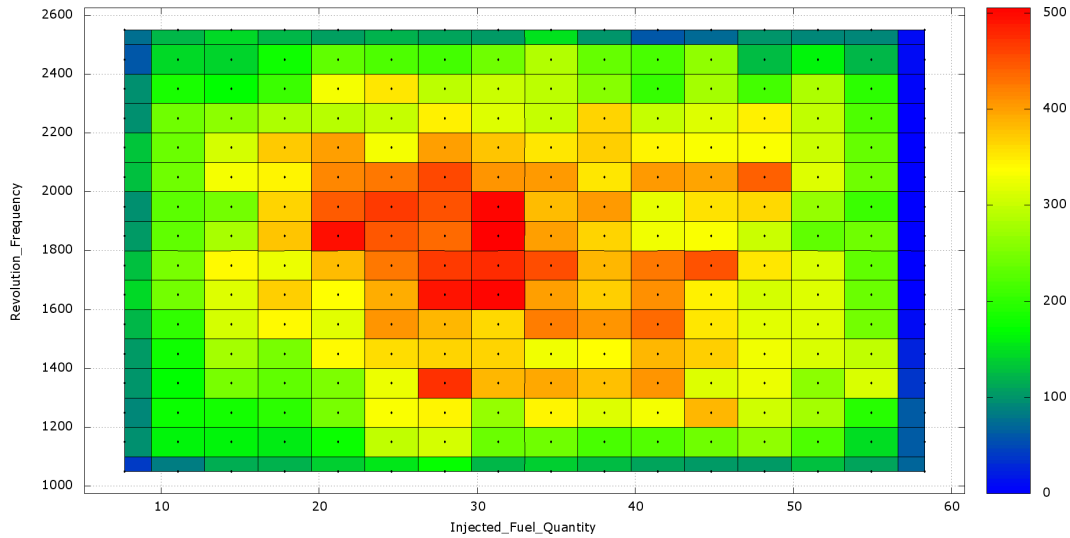
The data density has its maximum in the center of the graph. The border regions are weakly pronounced. However, the maximum counts per bin in Figure 49 is higher than in the weighted measurement run, i.e. 625 counts compared to 500. Furthermore, the maximum area is much smaller in the unweighted run, which points out, that the unweighted run causes a much more heterogeneous data density, than the weighted measurement run. Both data densities are not homogeneous over the whole data space range.

The reason for the pronounced peak of counts in Figure 49 is given by the measurement ramps itself. The probability of start- and end-points of a measurement ramp is equal for any destination in the data space, due to the uniformly distributed random numbers. However, the probability of vectors, which are given by the connection of a start- and an end-point, crossing the center of the data space is strongly increased. Starting a measurement ramp in



**Figure 49:** Histogram of data density (color scale) for an unweighted measurement run. Resolution of data space is 16 bins.

the border region of the data space, the end-point of the measurement has to be in on of the remaining 255 bins out of 256. In order not to cross the center region, the end-point has to be in reduced set of bins, which can be approximated to:  $2 \cdot 3 \cdot 16 - 3 \cdot 3 = 87$  bins. Any other bin, would result in a vector that crosses the center region. So, the probability is roughly 1 over 3 not to cross the middle of the data space, thus we expect, that the peak height is 3 times higher than the basis region. Roughly, this relation is shown in Figure 49.



**Figure 50:** Histogram of data density (color scale) for a weighted measurement run. Resolution of data space is 16 bins.

The weighted measurement run (see Figure 50) shows a less pronounced maximum peak, than the unweighted measurement run (see Figure 49). The more homogeneous distribution is caused by the weighting function of the random numbers that define the start- and end-points of the measurement ramps. Since the data density in the center increases rapidly,

---

the probability of a start- or end-point, which is in the center becomes improbable. So, the further gain of counts in the center is caused by passing measurement ramps from one border region to the opposite border region. In this case, less bins are available for a valid measurement route, thus the probability for passing measurement routes decreases. Finally, the data density becomes more homogeneous with weighted random numbers, but the distribution of data points is still not sufficient for further optimization processes.

---

### 5.1.2 Dijkstra Algorithm

---

The *Dynamic Test Drive* module has been upgraded with the *Dijkstra* algorithm. In general, the *Dijkstra* algorithm finds the shortest path between two vertices of a graph. The data density on each vertex is treated as additional time effort. So, the *Dijkstra* algorithm is able to avoid vertices with a higher data density. A more detailed description is given in Section 4.

#### 5.1.2.1 Environment

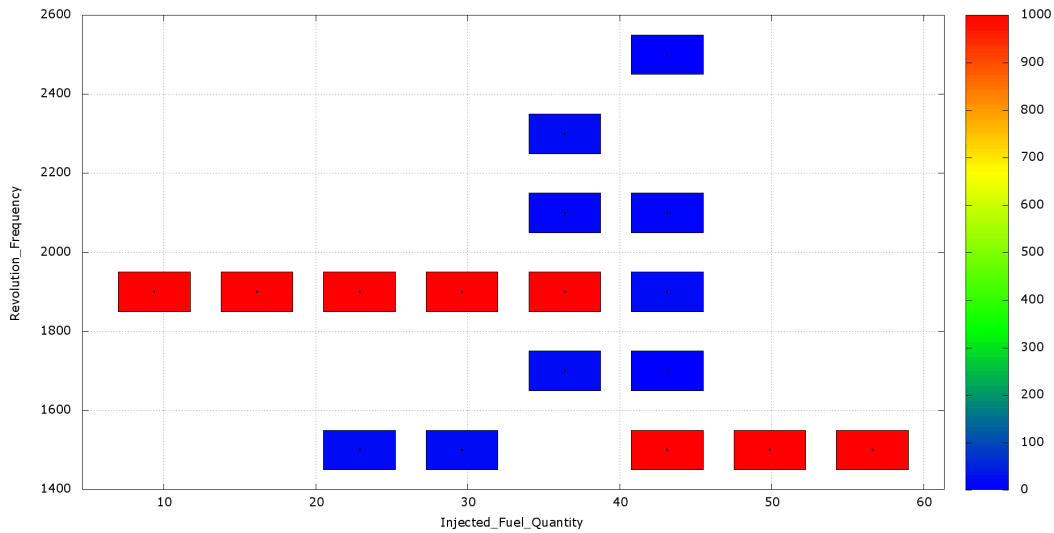
The environment of this method is similar to the environment of *Dynamic Test Drive*, but needs a graph of the connections between each adjacent leaves of the data tree *Tree*. The graph is given by the direct connections of leaves, which have to have at least one border in common.

#### 5.1.2.2 Assumption

This method routes between two given start- and end-points, which are derived by weighted random numbers. This route should prefer weakly occupied bins than ones with a high data density. So the data density should become much more homogeneous.

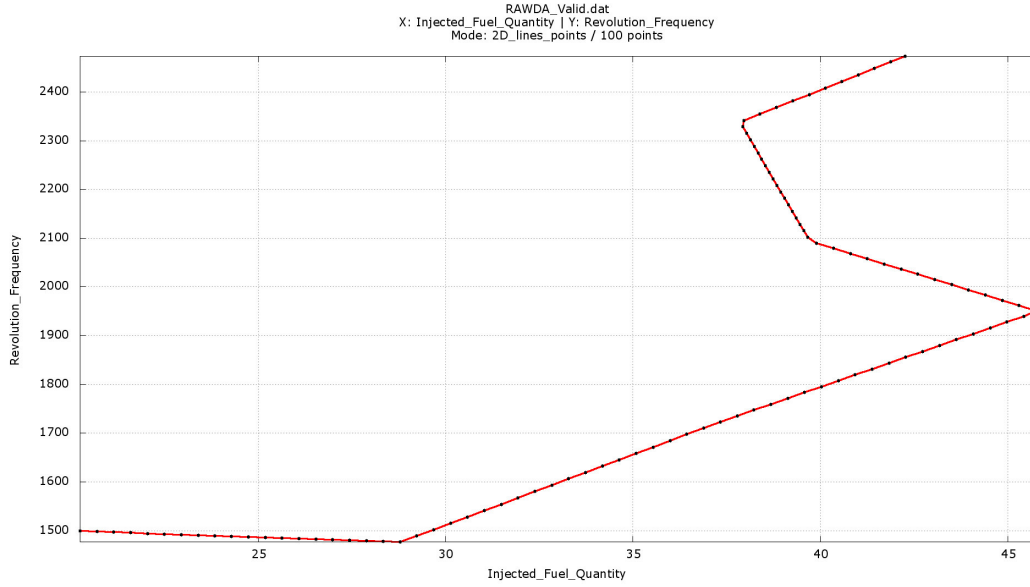
#### 5.1.2.3 Proof of Principle

This proof is split into two parts, the illustration of the *Dijkstra* routing, with respect to the given data density and secondly, the analysis of the data space coverage improvement due to the *Dijkstra* algorithm in Subsection 5.1.1.3.



**Figure 51:** Data space histogram with obstacles for the Dijkstra routing illustration. Red squares are artificial obstacles, which have to be bypassed by the Dijkstra algorithm. Blue squares mark the derived path.

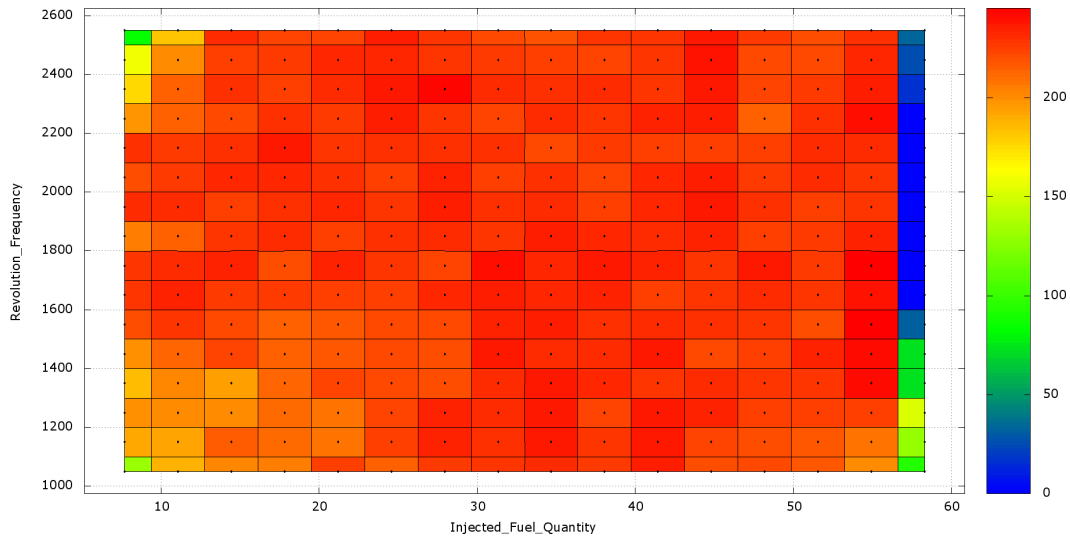
Figure 51 shows the artificial obstacles and the chosen path by the *Dijkstra* algorithm. Figure 52 shows the raw data points of the taken path. The routing is based on the resolution of *Tree* and the measurement frequency. One sees clearly, that the *Dijkstra* algorithm chooses a path, that avoids the obstacle. This ability has been included into the *Dynamic Test Drive*



**Figure 52:** Data points given by Dijkstra algorithm.

algorithm, in order to handle local inhomogeneities. Furthermore, the over-pronunciation of the center region of the data space should be reduced by this approach.

Figure 53 shows the data space histogram of a measurement run, with enabled weighting function and *Dijkstra* routing. The data density is much more homogeneous than in Figure 50. Every bin counts around 230 events. Only the left upper and lower corner (low injected fuel) and the region of high injected fuel shows a much lower frequency.

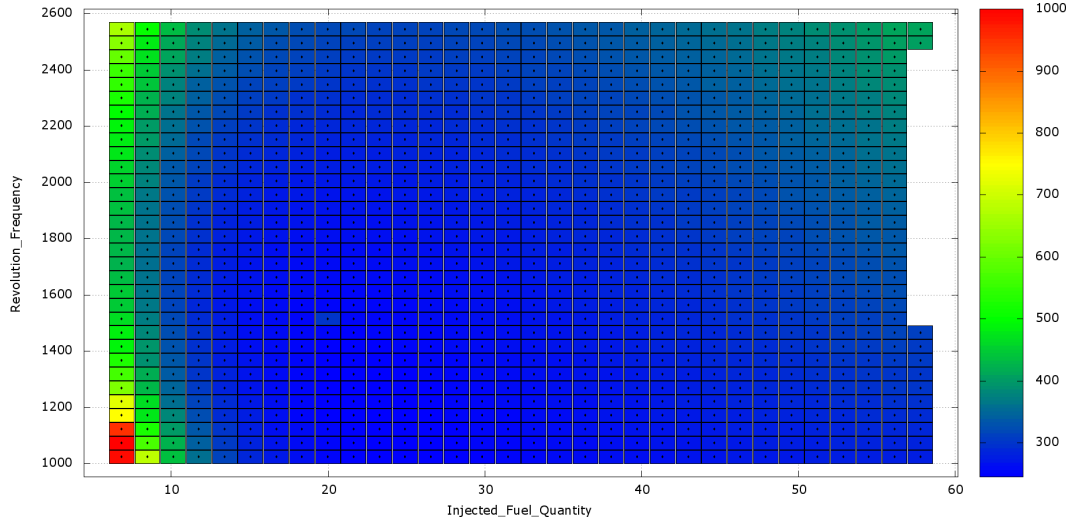


**Figure 53:** Histogram of data density for a weighted measurement run with Dijkstra routing. Resolution of data space is 16 bins.

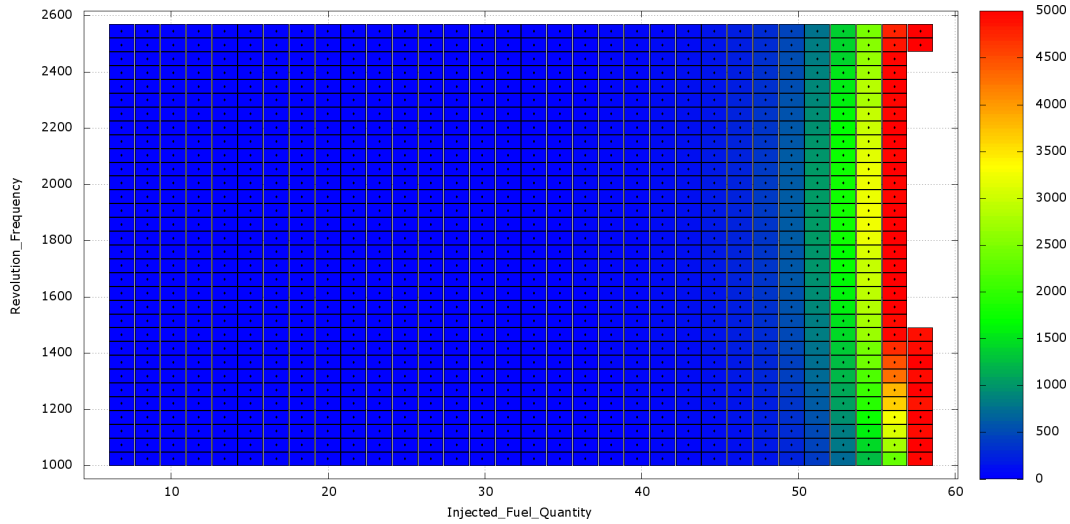
These inhomogeneities are caused by the hard sensor bounds, not by the *Dynamic Test Drive* algorithm. Filter functions monitor observables, which have been configured with a hard boundary condition. So measurement ramps has to be interrupted, in case of boundary



value violations. In this case the low revolution frequency, low injected fuel corner has been partially filtered due to the limitation of the specific fuel consumption (see Figure 54). The upper limit of the specific fuel consumption is given by  $1000 \frac{g}{kWh}$ . The high revolution frequency low injected fuel corner has been filtered due to a low limit violation of the engine torque observable. The minimal value of torque has been configured with  $2Nm$ . Finally, the high injected fuel region around  $58 \frac{mm^3}{cycle}$  has been filtered due to the maximal smoke value definition. The high value of smoke has been set up to 5000.



**Figure 54:** Maximal fuel consumption values with respect to data space bins.



**Figure 55:** Maximal smoke emission values with respect to data space bins.

---

### 5.1.3 Data Cleaning

---

The raw data points are cleaned by boundary filters and a linear regression method. The detailed description is given in Subsection 4.2.4.

#### 5.1.3.1 Environment

Raw data points are grouped in measurement sweeps. A complete sweep is transferred from the test bench/data model to the data analysis algorithm. The measurement frequency is configured by the user, typically the measurement frequency is between 5 and 10 Hz.

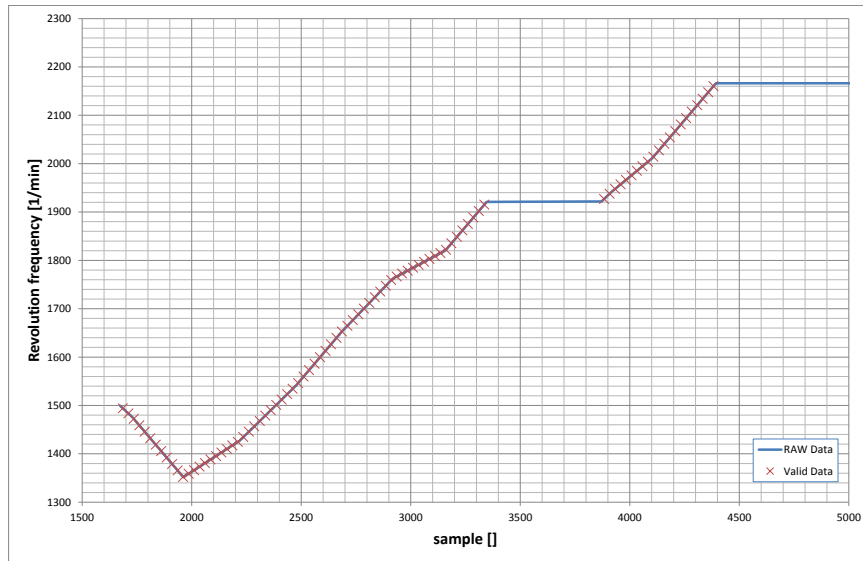
#### 5.1.3.2 Assumption

The number of data points should decrease, in order to reduce the run-time of complex optimization algorithms. Furthermore, the signal quality of observables should be improved, additionally, the statistical and systematical error bar should be evaluated.

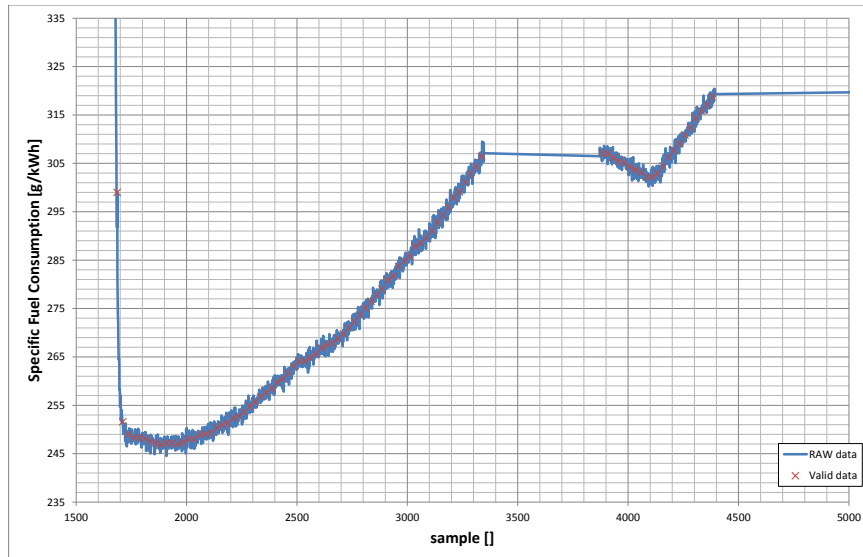
#### 5.1.3.3 Proof of Principle

The data cleaning algorithm differentiates between actuators and observable/result values, actuators are values, which are set, therefore these values are exact without statistical or systematical errors. Observables are measured by the test bench system, so the errorbar depend on the physical system and quality of the measurement system.

Figure 56 shows the raw data stream of a measurement sweep in blue. The red crosses show the interpolated data points. Each red cross is exactly on the blue raw data points. The detection frequency of valid data points depend on the relative change of observables of interest. The horizontal line between two sweeps is artificially increased, in order to improve visual differentiation between two sweeps. Figure 57 shows raw and valid data points of the same sweeps, but for the specific fuel consumption, which is an observable of interest. One sees clearly the scattering of the raw data points, the valid data points are derived by a moving linear regression function. The errorbar of each valid data point is given by the maximal derivation of the data points of an interpolation of its linear regression function. The detection frequency of valid data points depend on the user configuration, a minimal detection frequency and a maximal relative change of observables. A detailed explanation is given in Subsection 4.2.4.



**Figure 56:** Raw data points (blue) of measurement sweep with interpolated valid data points (red) of dynamic actuator revolution frequency.



**Figure 57:** Raw data points (blue) of measurement sweep with interpolated valid data points (red) of observable specific fuel consumption.

---

## 5.1.4 Controller Oscillation Filter

---

The *Controller Oscillation Filter* has been applied in order to detect and filter controller oscillations (see Subsection 4.2.4.2).

### 5.1.4.1 Environment

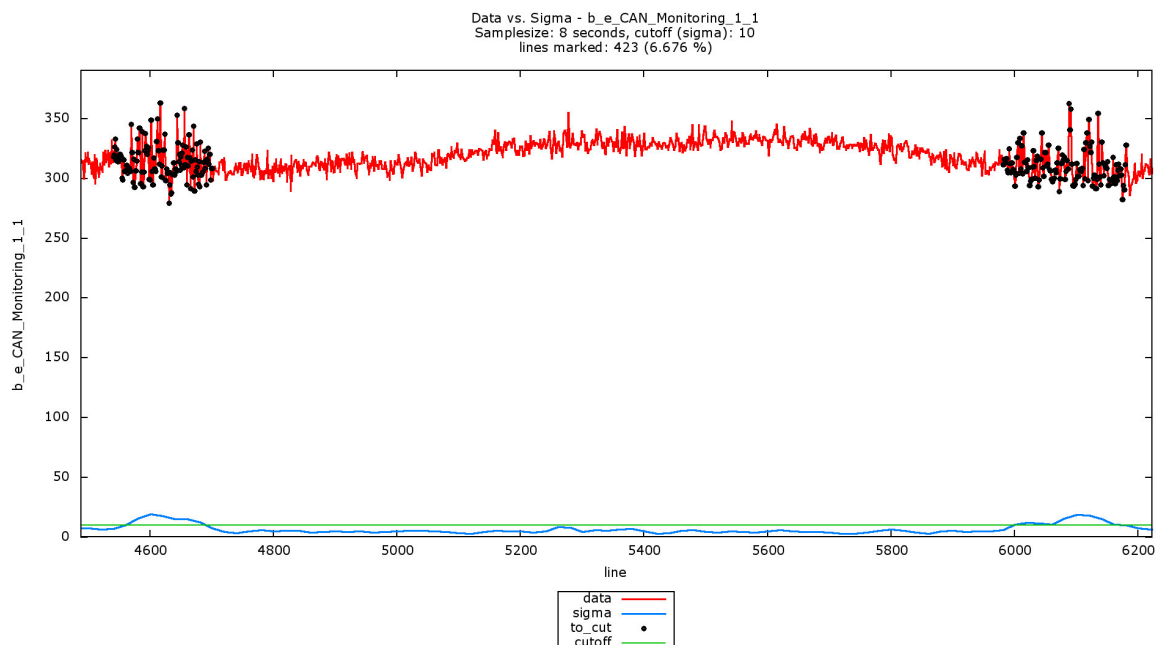
The raw data stream is processed by *b.osco* during the measurement. By deriving and comparing the standard deviation  $\sigma$  from the data points to a moving linear regression, *b.osco* marks and filters controller oscillations.

### 5.1.4.2 Assumption

*b.osco* should detect controller oscillations. The definition of controller oscillations are done by the user, by defining an upper threshold for  $\sigma$ . Other data points should not be affected by this method.

### 5.1.4.3 Proof of Principle

Figure 58 shows a typical oscillation of the throttle/air valve actuator. The marked data points are set to invalid. Figure 58 shows three graphs, raw data points (red) with marked data lines (black circles), the moving standard deviation (blue) and the deviation cutoff value (green). If the moving standard deviation is greater than the configured cutoff, the raw data points are marked. The method marks the oscillating data points, detected in the specific fuel consumption. Other data points are not affected.



**Figure 58:** Controller oscillation detection with *b.osco*

---

## 5.1.5 Measurement Specialization

---

The measurement specialization is necessary to adopt to the structure of the optimum and in order to reduce measurement time. The validation of this method is separated into two passages, the validation of the local model and the validation of the sensitivity analysis. This method is explained in Subsection 4.2.5.

### 5.1.5.1 Environment

This part of the algorithm modifies the data tree *Tree*, defined in Subsection 4.2.1. The refinement process analyses the improvement of the constrained solution, moreover the algorithm compares newly measured valid data points to local polynomial data models. In order to visualize the results of this validation, we apply these methods to a problem with two actuators, only.

### 5.1.5.2 Assumption

This algorithm should refine the data tree corresponding to the potential of improvement of the unconstrained solution. Additionally, the refinement should mark passages of the data space, which are not covered sufficiently. Finally, this algorithm calculates the impact of each single actuator on the solution improvement, so that an asymmetric split can be performed. The run-time and memory advantages of asymmetric instead of symmetric splits should also be proofed.

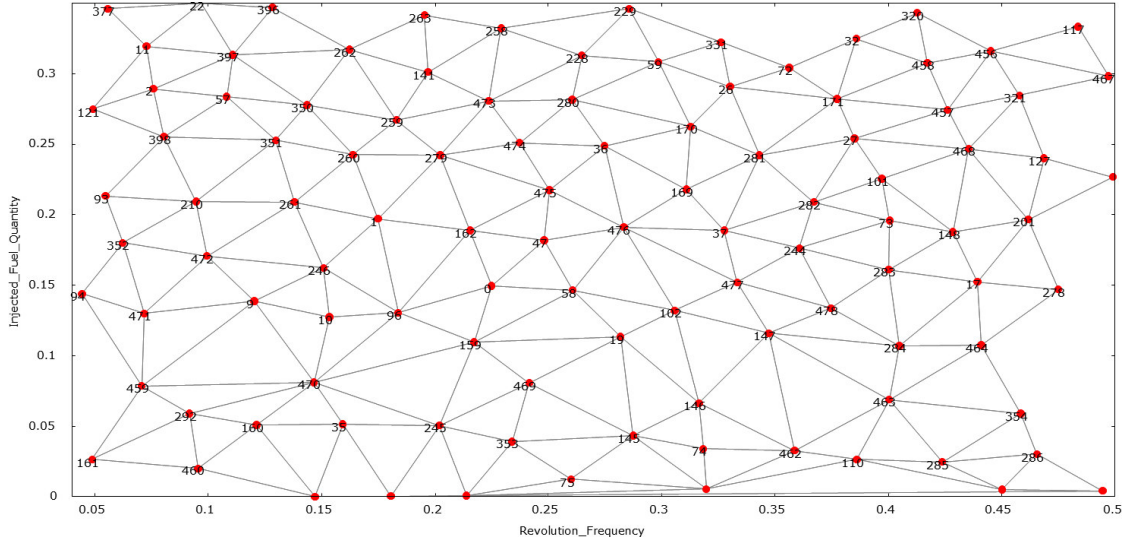
### 5.1.5.3 Proof of Principle

**Polynomial fitting method:** The first part of the proof deals with the detection of data space regions, which are not well represented by the current amount of data. Therefore, *AmmOC* is able to do an analysis of each data point as described in Paragraph 4.2.5.1. In this analysis *AmmOC* tries to derive each data point by its nearest neighbor data points. If the derivation is larger, than the user defined threshold, e.g. 5 %, the corresponding bin of the data tree *Tree* has to be split in each dimension.

The first step for this analysis is the determination of the nearest neighbors of each data point (see Subsection 4.4.1 for further information). Figure 59 shows the Delaunay triangulation of an example measurement sample of the region around the sample point. The next analysis step is the detection of regions, which are not well represented by a polynomial fit function. In this case all data points can be fitted by the nearest neighbor data points with a derivation  $\leq 1\%$ . Figure 60 shows the nearest neighbors of a sample data point, the fill color of the Voronoi cells represent the value of specific fuel consumption. The data point is given in relative coordinates for each dynamic actuator. The static actuators are set to default values. The result sensor values are given in absolute coordinates. The sample points, which have to be checked by the polynomial function is located on Revolution Frequency =  $0.244 \frac{1}{min}$ , Injected Fuel Quantity =  $0.149 \frac{kg}{cycle}$ . The current specific fuel consumption is  $264.867 \frac{g}{kWh}$ .

The evaluation of the Levenberg-Marquardt-algorithm derives the fitted specific fuel consumption value to  $265.112 \frac{g}{kWh}$ , which is in good agreement with the measured value. In this example, we show, that the method works well for the validation of newly measured data points. Especially, the selection of nearest neighbor data points help to improve the creation of local models. In general, all data points are easy to reproduce for less than three dynamic

actuators, therefore, the deviation versus number of actuators has to be studied separately in the next passage.



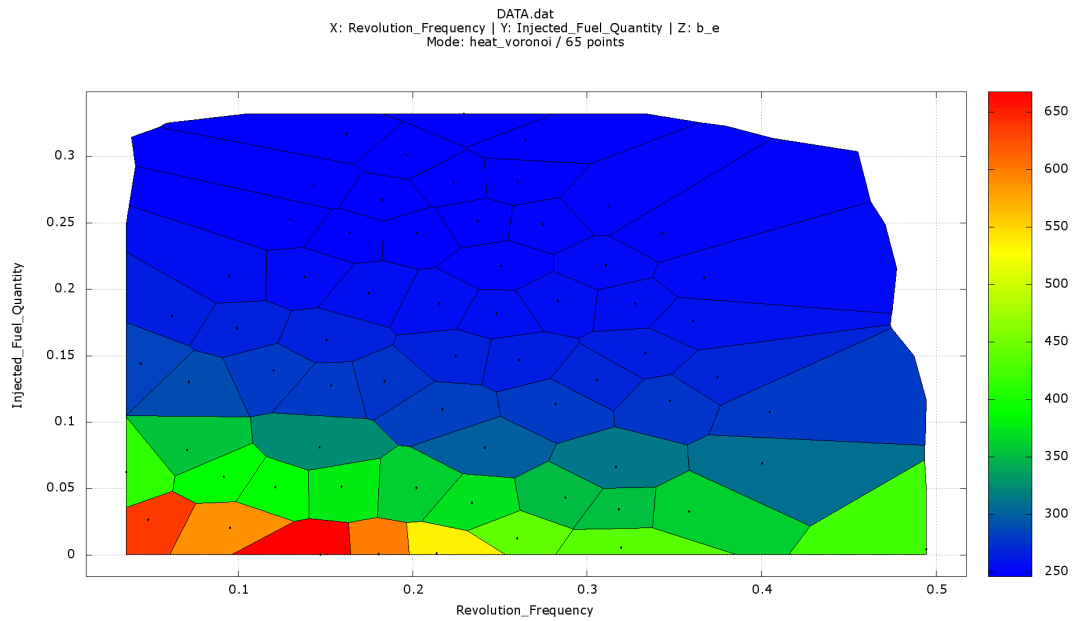
**Figure 59:** Delaunay triangulation excerpt of measurement example in preparation of measurement specialization. Delaunay connections, which are outside the filtered range, are blanked out.

#### DEVELOPMENT OF THE AVERAGE ERROR DEPENDING ON THE DIMENSION OF DATA SPACE

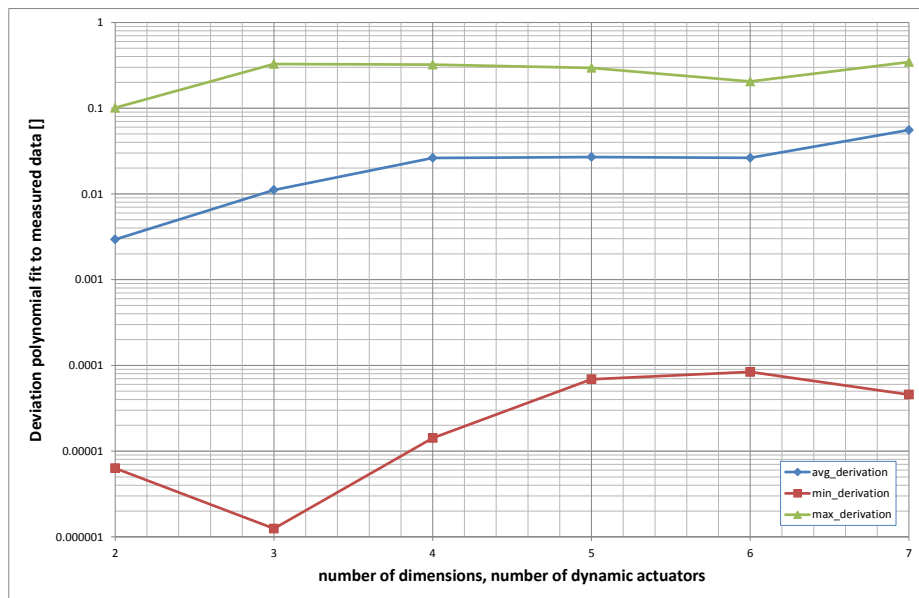
In this insertion, we study the influence of the dimension, i.e. number of dynamic actuators, on the creation of the local models. The number of valid data points is fixed, the number of dynamic actuators is increased. The number of data points, which participate in the local fit is given by  $10 \cdot (d + 1)^2$ , where  $d$  is the number of dynamic actuators. So, it is possible to observe the development of the average errorbar of the polynomial fit function depending on the dimension of the data space.

Figure 61 shows the development of the average, minimal and maximal derivation from the polynomial fit to the measured data for different numbers of dynamic actuators. The average derivation is extremely small for two and three dynamic actuators, due to the small number of degrees of freedom of the polynomial fit. For a larger number of dynamic actuators, the average deviation increases. The main reason for this behavior is the increasing number of degrees of freedom of the polynomial fit, secondly, the number of data points is fixed to 5000. That is why the coverage of the data space is poor for higher number of actuators. So, a polynomial fit has to include data points of regions, which does not represent the region of the sample point. Consequently, the algorithm has to improve the resolution of the data tree and furthermore, the algorithm has to continue to measure data points in order to reduce the mean error in this region.

**Sensitivity Analysis and Asymmetric Split Method:** The advantage of asymmetric splits is the reduced number of bins in the data tree, thus the reduced run-time and memory consumption in more complex algorithms, e.g. *Dijkstra* routing. This method is based on the analysis of the improvement of the unconstrained solution. Therefore, the algorithm determines the impact of each dynamic actuator on the improvement of the objective function. Therefore, *AmmOC* has to perform a sensitivity analysis before the splitting process begins. Table 7 shows the three partial sweeps, in order to determine the actuator that cause the major improvement of the specified criterion. In this example the unconstrained solution

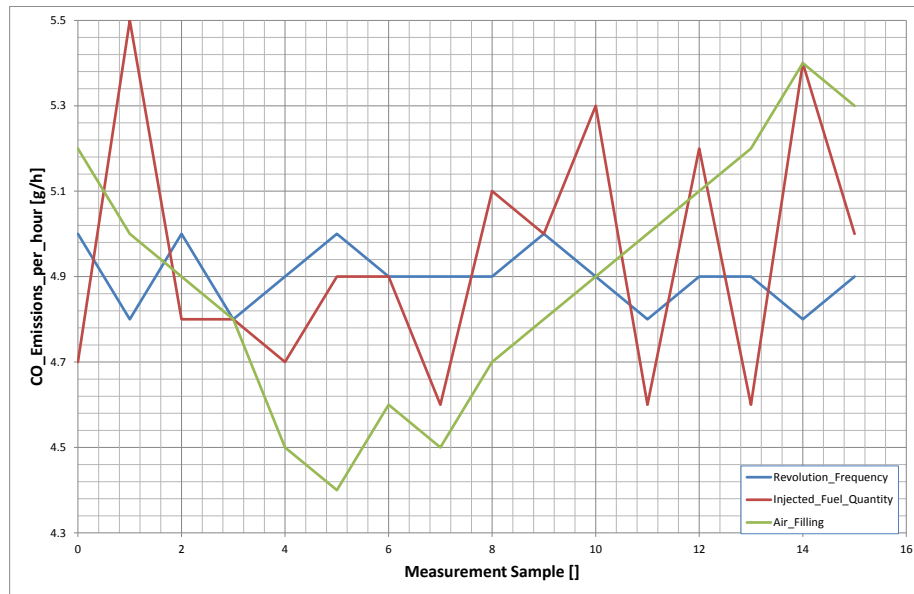


**Figure 60:** Nearest neighbors of sample point ([0.224,0.149,264.867]) for an example with three dynamic actuators



**Figure 61:** Dependency of deviation of polynomial fit to measured data in accordance to the number of actuators

improves in the category *CO\_Emissions\_per\_hour*. The sensitivity analysis shows, that the mayor impact is due to the 'Air\_Filling' actuator. Figure 62 shows the *CO* value during the sensitivity analysis divided for the different dynamic actuators. Each function shows the absolute value of *CO\_Emissions\_per\_hour* during the variation of each single actuator. The smallest value of all functions marks a local optimum of this criterion in this region of the data space. Consequently, the corresponding bin has to be split only in the 'Air\_Filling' dimension. The borders of 'Revolution\_Frequency' and 'Injected\_Fuel\_Quantity' do not have to be affected.



**Figure 62:** Visualization of measurement sweeps of sensitivity analysis



**Table 7: Measurement sweeps for sensitivity analysis**

Vary Revolution\_Frequency ... 1379.00690257 +/- 50.0( [1300.0, 1400.0] )

Revolution Frequency	Injected Fuel Quantity	Air Filling	be	CO	HC	NOx
1405.3	26.3	297.7	250.1	5	1.7	6.1
1338.6	26.3	297.7	249.2	4.8	1.6	6.4
1418.6	26.3	297.7	250.5	5	1.7	6
1351.9	26.3	297.7	248.8	4.8	1.6	6.4
1372.6	26.3	297.7	257.6	4.9	1.6	6.2
1426.3	26.3	297.7	250.7	5	1.7	6
1365.3	26.3	297.7	249.1	4.9	1.6	6.3
1359.3	26.3	297.7	249.8	4.9	1.6	6.3
1379	26.3	297.7	249.2	4.9	1.6	6.2
1413	26.3	297.7	250.5	5	1.7	6
1378.6	26.3	297.7	249.6	4.9	1.6	6.2
1345.9	26.3	297.7	249	4.8	1.6	6.4
1399.7	26.3	297.7	250	4.9	1.7	6.1
1391.9	26.3	297.7	249.7	4.9	1.7	6.1
1332.6	26.3	297.7	248.9	4.8	1.5	6.5
1386.3	26.3	297.7	249.9	4.9	1.6	6.2

Vary Injected\_Fuel\_Quantity ... 26.3108967061 +/- 1.6875( [26.25, 29.625] )

1379	25.4	297.7	248.8	4.7	1.7	6.6
1379	27.9	297.7	251.3	5.5	1.6	5.5
1379	26.1	297.7	249.4	4.8	1.6	6.3
1379	25.8	297.7	248.9	4.8	1.6	6.4
1379	25.6	297.7	249.3	4.7	1.6	6.5
1379	26.3	297.7	249.5	4.9	1.6	6.2
1379	26.3	297.7	250.6	4.9	1.6	6.2
1379	25.2	297.7	248.9	4.6	1.7	6.7
1379	27	297.7	250.9	5.1	1.6	5.9
1379	26.7	297.7	249.9	5	1.6	6
1379	27.5	297.7	250.8	5.3	1.6	5.7
1379	24.7	297.7	248.1	4.6	1.7	6.9
1379	27.2	297.7	250.4	5.2	1.6	5.8
1379	24.9	297.7	248.3	4.6	1.7	6.8
1379	26.6	297.7	250	5	1.6	6.1

Vary Air\_Filling ... 297.744745729 +/- 22.375( [275.0, 319.75] )

1379	26.3	285.6	249.6	5.2	1.6	4.9
1379	26.3	291.6	249.5	5	1.6	5.5
1379	26.3	297.6	249.7	4.9	1.6	6.2
1379	26.3	303.5	249.4	4.8	1.6	6.9
1379	26.3	315.5	248.9	4.5	1.6	8.6
1379	26.3	318.9	249.5	4.4	1.7	9
1379	26.3	309.5	249.8	4.6	1.6	7.7
1379	26.3	313	249.9	4.5	1.6	8.2
1379	26.3	307	249.6	4.7	1.6	7.4
1379	26.3	301	249.7	4.8	1.6	6.6
1379	26.3	297.7	249.7	4.9	1.6	6.2
1379	26.3	294.9	249.7	5	1.6	5.9
1379	26.3	288.9	249.8	5.1	1.6	5.3
1379	26.3	282.9	249.7	5.2	1.6	4.7
1379	26.3	279.7	249.7	5.3	1.6	4.4

---

## 5.1.6 Adaptive Space Compressor

---

The adaptive space compressor *ASC* method has been established, in order to reduce the number of valid data points for more complex analysis algorithms, by analyzing the spacial distribution of the data density. Furthermore, the algorithm improves the spacial homogeneity in accordance with the structure of the data tree. *ASC* filters less data points in regions, which have been refined several times. In this validation, we discuss the homogeneous case, thus we assume an equally split data tree.

### 5.1.6.1 Environment

This method works with the valid data points, which are already filtered by the time series compressor *TSC* method (see Subsection 4.3.1). In contrast to the *TSC* method, the *ASC* method is not based on the chronological appearance of the data points, but on the spacial distribution in the data space (see Subsection 4.2.1 and Subsection 4.3). Furthermore, this method has to be configured by the user. The configuration parameters are given by the requested number of filtered data points or the *ASC* length scale (see Equation (95)).

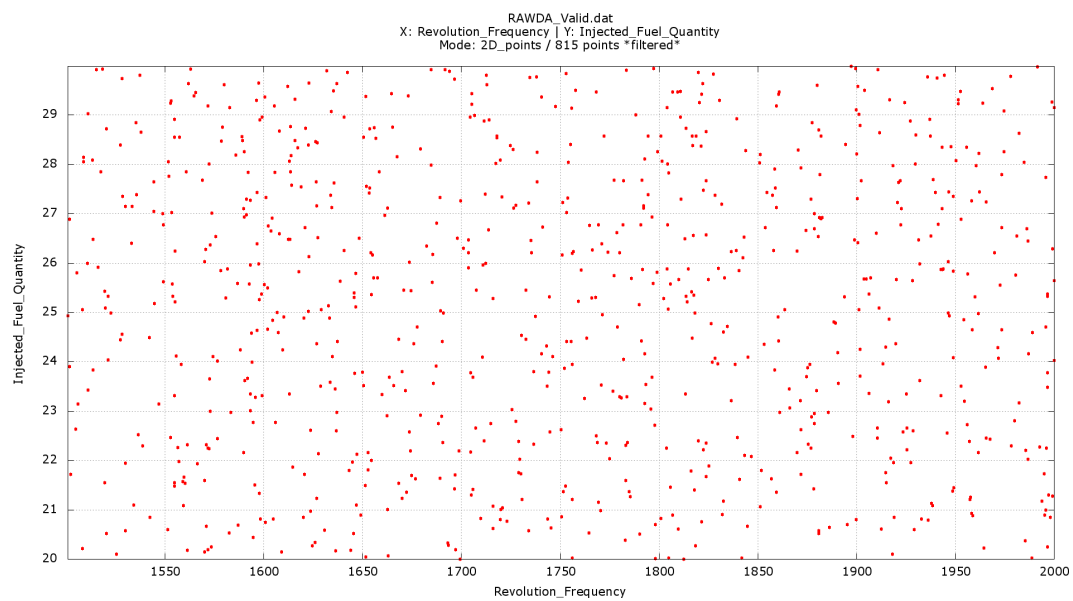
### 5.1.6.2 Assumption

This method has to increase the homogeneity of the data space, moreover this method should take into account the split structure of the data tree *Tree* (see Equation (95)). Finally, the calculated standard deviation within an hyper ellipsoid has to stay below an user defined value otherwise this method causes a warning message for the user.

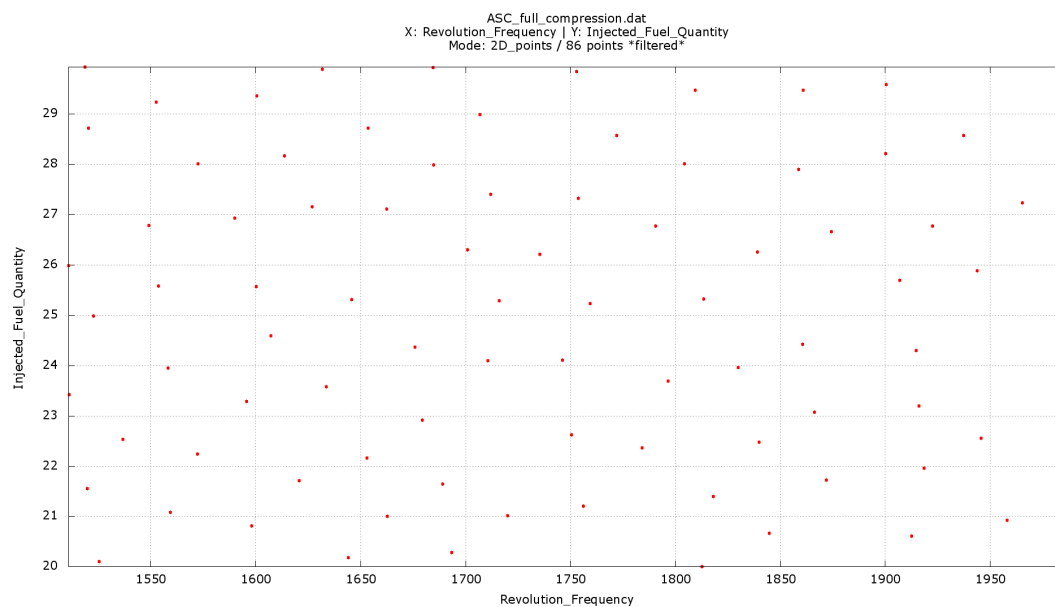
### 5.1.6.3 Proof of Principle

Figure 63 shows a typical data distribution for a two dimensional test case. In this test case, *Revolution Frequency* and *Injected Fuel Quantity* has been varied, during the measurement, the raw data points have been interpolated by the linear regression method (see Subsection 4.2.4). Additionally, the valid data points pass the *TSC* method.

Figure 63 shows, that the data coverage of this region is nearly complete and homogeneous. Figure 64 shows the result of the *ASC* method on the data distribution. The number of data points has been reduced from 815 valid data points to 86 filtered data points. All filtered data points are equally distributed, by filtering all other data points.



**Figure 63:** Sample region with valid data points



**Figure 64:** Sample region with adaptive space compressed data points

---

### 5.1.7 Determination of nearest Neighbors

---

The determination of the nearest neighbors of a vertex in a given graph is essential for two methods of this work, i.e. the tree refinement due to unexpected result values (see Subsection 4.2.5.1) and the creation of the optimization network for the evolutionary optimization algorithm (see Subsection 3.2.3). The theoretical definition is given in Subsection 4.4.1.

#### 5.1.7.1 Environment

In general, the algorithms for the determination of nearest neighbors can be applied on raw, valid and ASC filtered data sets. Typically, the Voronoi tessellation is applied on ASC filtered data sets, only. ASC filtered data points are equally distributed in data space, in accordance with the split level of the corresponding bins of *Tree*, which is given by the data tree of all valid data points (see Subsection 4.2.1).

#### 5.1.7.2 Assumption

In order to solve this problem we apply two different algorithms, i.e. the *Recursive Light Ray Algorithm* (see Subsection 4.4.1.2) and the state of the art method *Beneath and Beyond* (see Subsection 2.3.2) of the program *Polymake* [Jos02]. Both algorithms are based on very different theoretical concepts, however both algorithms should deliver the same nearest neighbor information of a given data point distribution. Furthermore, the *Recursive Light Ray Algorithm* supports an iterative mode. In this mode, it is possible to add new data points to an existing Delaunay triangulation. The resulting triangulation should be equal to an directly produced triangulation.

#### 5.1.7.3 Proof of Principle

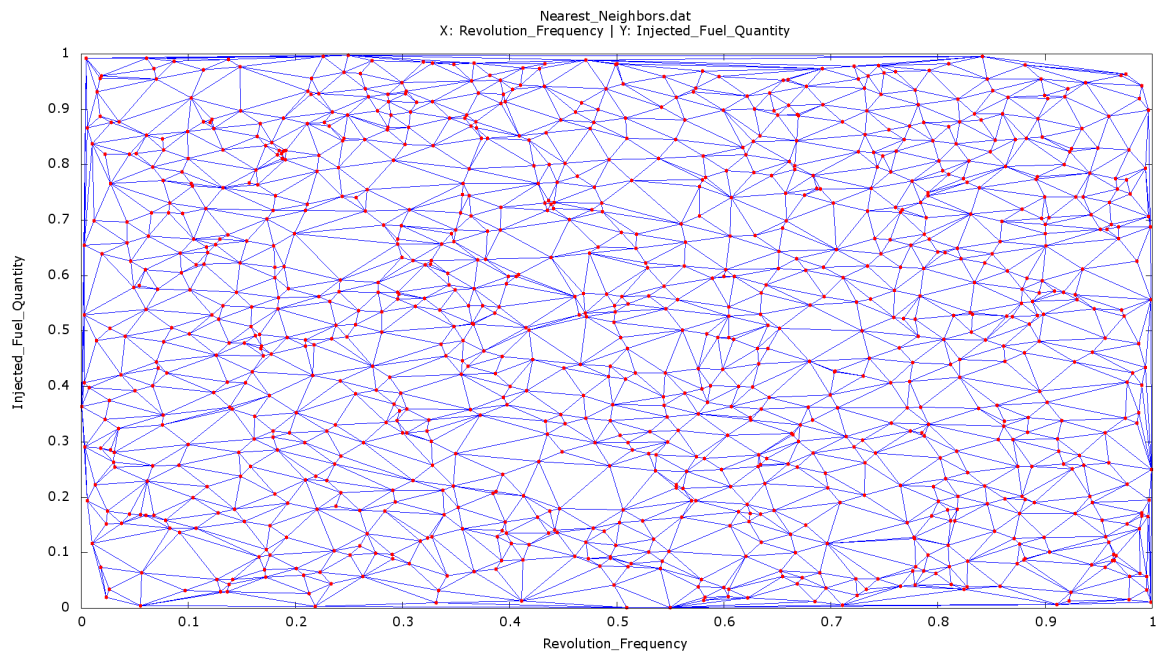
**Comparison of *Polymake* and *Recursive Light Ray Algorithm*:** Figure 65 and Figure 66 show the Delaunay triangulation computed with *Polymake* and *Recursive Light Ray Algorithm*. Both triangulation illustrations are equal.

The recursion depth of the *Recursive Light Ray Algorithm* method is limited to 50 iteration steps. The algorithm supports an option to exclude data points, with a distance greater than a given threshold. In this run, we set this option to a maximal value.

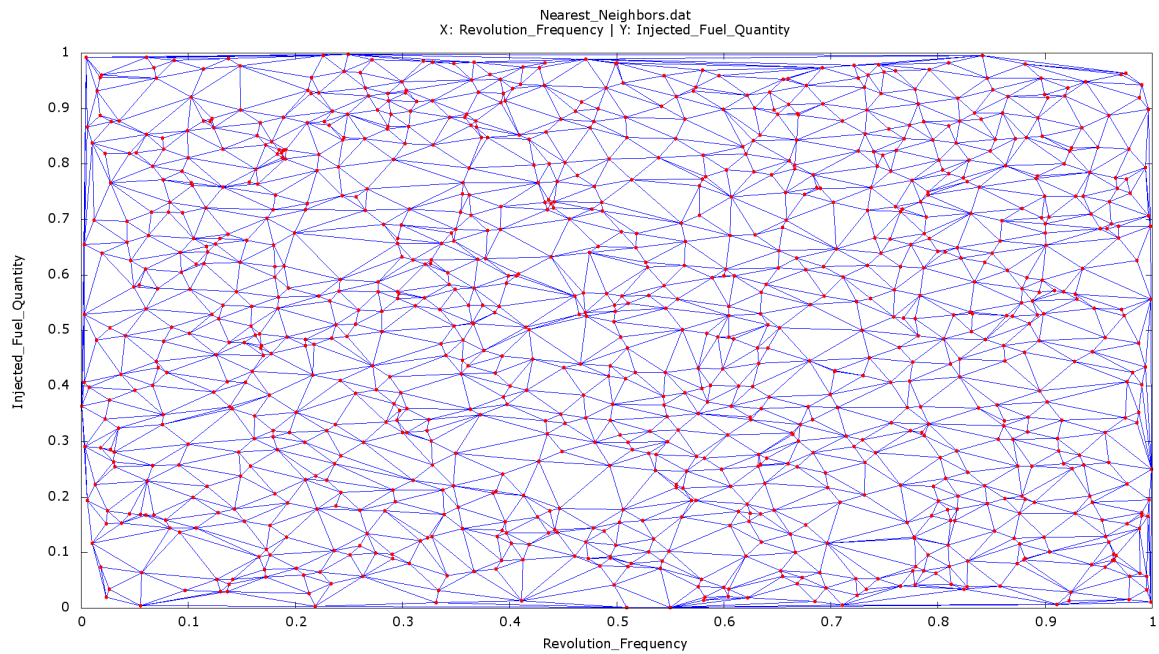
Figure 67 does not show any missing or more connections between data points, so the *Recursive Light Ray Algorithm* does not find more or less connections than the *Polymake* algorithm. Consequently, the *Recursive Light Ray Algorithm* does not predict wrong connections.

In general, both algorithm can be applied for the determination of the nearest neighbors of a given data point distribution. In case of the derivation of the optimization graph for the *Greedy Ants* algorithm, we apply the *Polymake* algorithm. For the specialization process (see Subsection 4.2.5) we apply the *RLR* method. In the following benchmark subsection, we extend the validation process to higher dimensions and more data points (see Subsection 5.2.1).

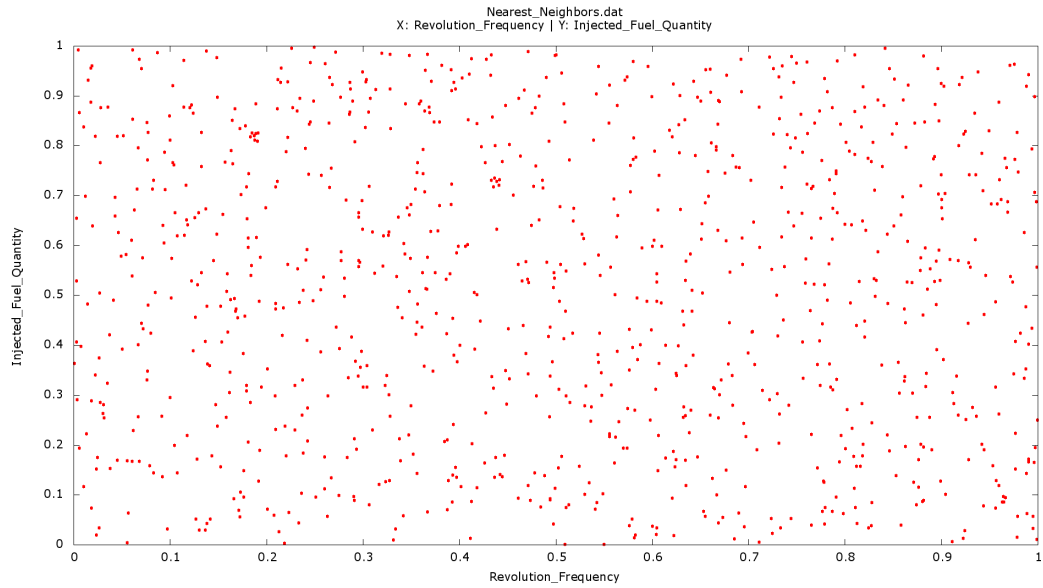
**Validation of iterative mode of *Recursive Light Ray Algorithm*:** The iterative mode of *Recursive Light Ray Algorithm* is able to add new data points to an existing Delaunay triangulation. Figure 68 shows an existing Delaunay triangulation of 50 data points. In the next iteration step, 50 new data points are added to the triangulation. First, *Recursive Light Ray Algorithm* determines the nearest neighbors of the 50 new data points in combination with the existing data points (see Figure 69). In this special test case the incomplete triangulation is nearly as big as the final triangulation. During a real measurement, the location of



**Figure 65:** Delaunay triangulation of a small test sample determined with *Polymake*

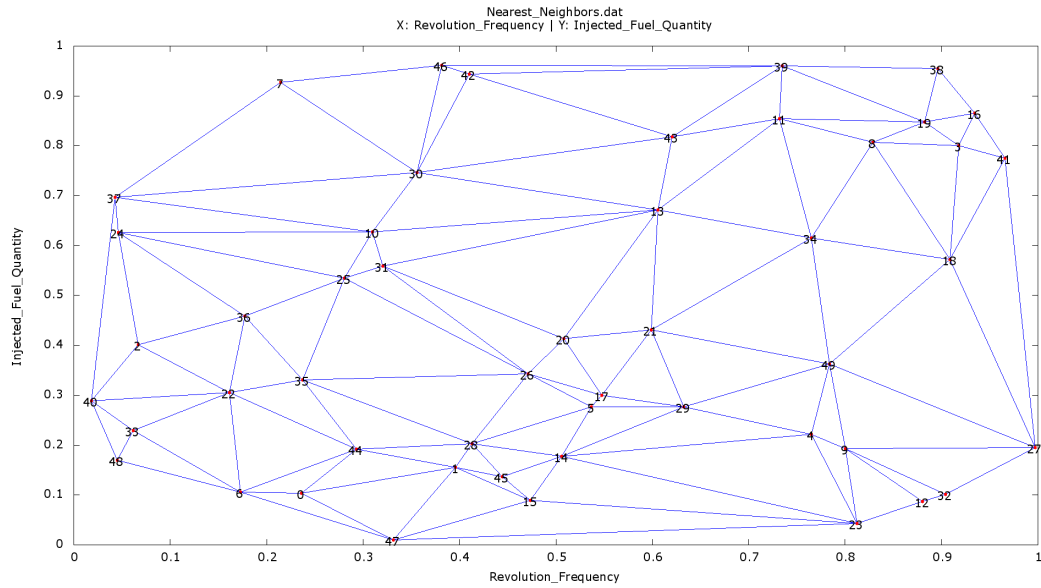


**Figure 66:** Delaunay triangulation of a small test sample determined with *Recursive Light Ray Algorithm*

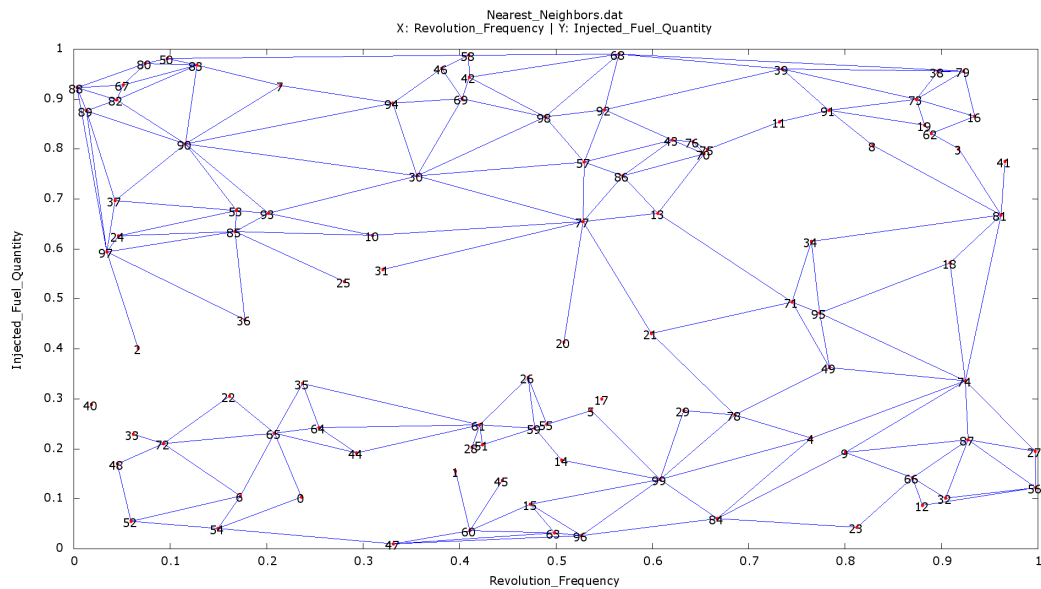


**Figure 67:** Difference of the results of RLR and *Polymake*

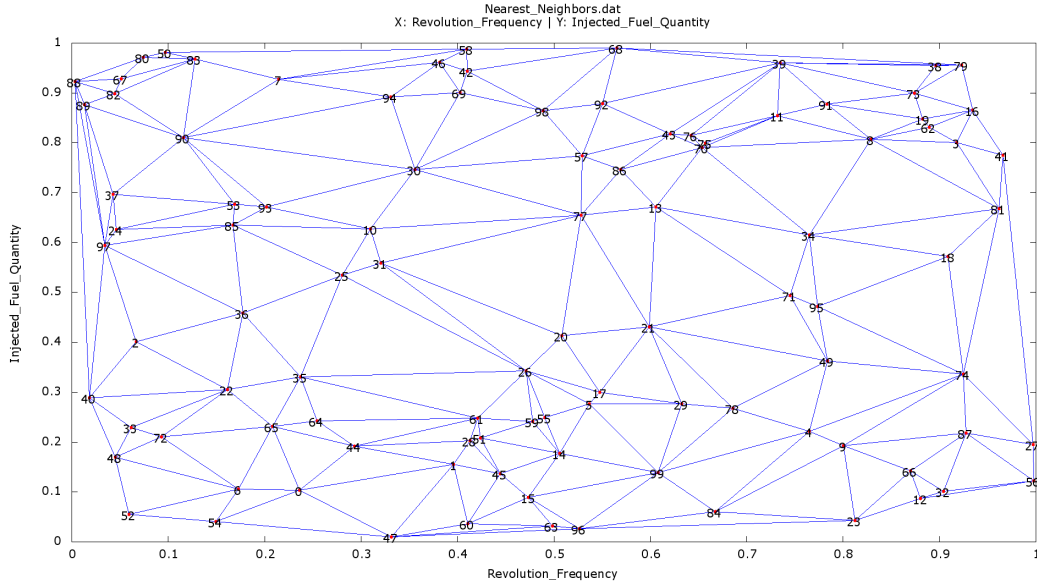
the new data points is strongly limited to a very small volume of the data space, therefore, the incomplete triangulation is much smaller, than the final triangulation. That is why the iterative mode saves a lot of run-time. In the final step, the algorithm updates every vertex, which is member of the incomplete triangulation and member of the set of old data points. So, the final triangulation is completed.



**Figure 68:** Delaunay triangulation of a small test sample



**Figure 69:** New partial Delaunay triangulation (incomplete)



**Figure 70:** New Delaunay triangulation append to existing net

### 5.1.8 Determination of Tolerance Table

The determination of the tolerance table is necessary to formulate the maximal gradient constraints (see Equation (8)) and in order to determine the network graph for the evolutionary algorithm *Greedy Ants*. The implementation is described in Subsection 4.4.2.

#### 5.1.8.1 Environment

The tolerance table is derived from the points that take part in the optimization process. These points are marked as valid. So, these points have been filtered by the time series compressor and the adaptive space compressor. The definition of the participants and its maximal gradients in the constrained solution is done by the user (see Equation (8)). Due to the tolerance table, *AmmOC* is able to formulate the tolerance constraints for the integer program and derive the network graph for the evolutionary algorithm *Greedy Ants*.

#### 5.1.8.2 Assumption

This method should derive all data points, which maintain the tolerance constraint to a data point  $d_i \in D$ . The result is stored as matrix  $T_{ij} \in \mathbb{B}^{n \times n}$  (see Equation (115)) where  $n$  is the number of filtered data points.

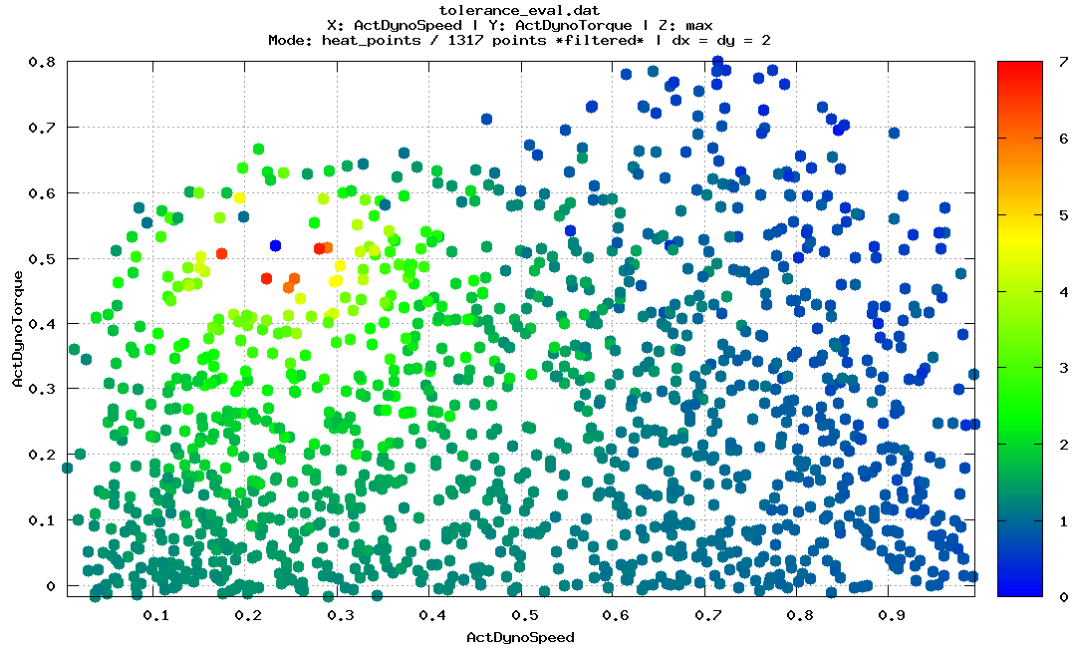
#### 5.1.8.3 Proof of Principle

In order to proof the principle of the determination of tolerance tables, we compute a small sample run. This run has three actuators, revolution frequency (*ActDynoSpeed*), engine torque (*ActDynoTorque*) and the air valve angle (*Alpha*). The solution surface or manifold  $S$  is given by the revolution frequency and the engine torque. Together with air valve angle, they define the whole data space. Result values should be neglected in this validation.

Figure 71 shows all data points, which conserve the tolerance constraint relating to data point  $d_i$  at (0.24/0.515) (blue point). The z-column in this illustration is given by the gradient (see Equation (114)) value of the data point. The user defined upper bound for the



gradient of  $\alpha$  has been set to 7. One notices, that data points in the direct environment around the data point  $d_i$  has been erased. Furthermore, the gradient value (z-column) decreases proportionally to the distance to the data point  $d_i$ .



**Figure 71:** Evaluation of tolerance table. Remaining data points, after chosen data point 0 with maximal gradient shown in color.

---

## 5.2 Benchmarks

---

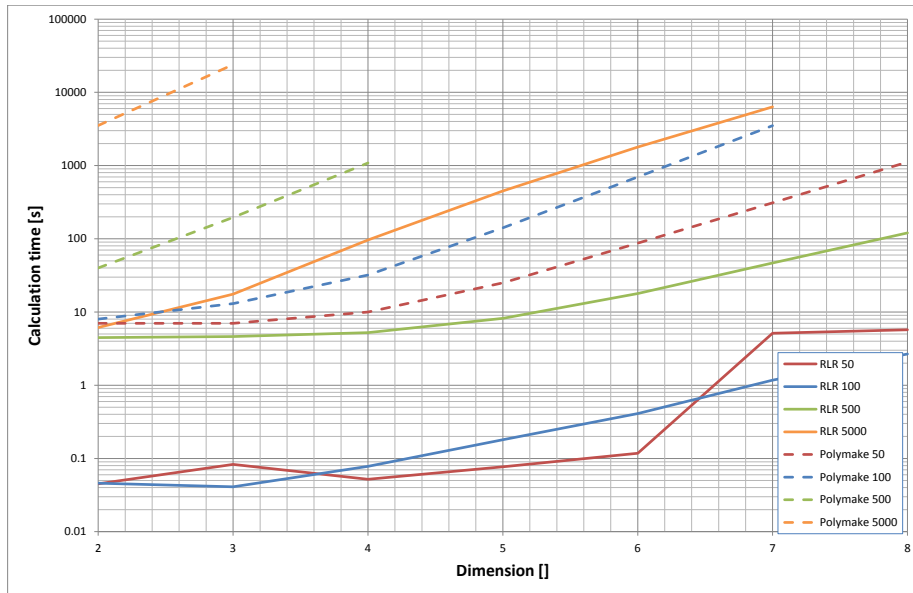
### 5.2.1 Voronoi Tessellation

---

#### 5.2.1.1 Comparison of Run-Times between Recursive Light Ray and Polymake Algorithm

Besides the correctness of the nearest neighbor determination, the run-time of the algorithm is very important. Therefore, the performance of the Recursive Light Ray *RLR* and the *Polymake* Algorithm has to be compared.

Figure 72 shows a comparison between *RLR* algorithm and *Polymake*. It shows the average calculation time to determine the Voronoi Tessellation for different number of data points and increasing number of dimensions. The run-time of both algorithms depend strongly on the number of data points and on the number of dimensions. The *RLR* algorithm is up to two orders of magnitude faster than *Polymake*. Furthermore, the *RLR* algorithm supports an iterative append mode, which will be studied in the next paragraph.



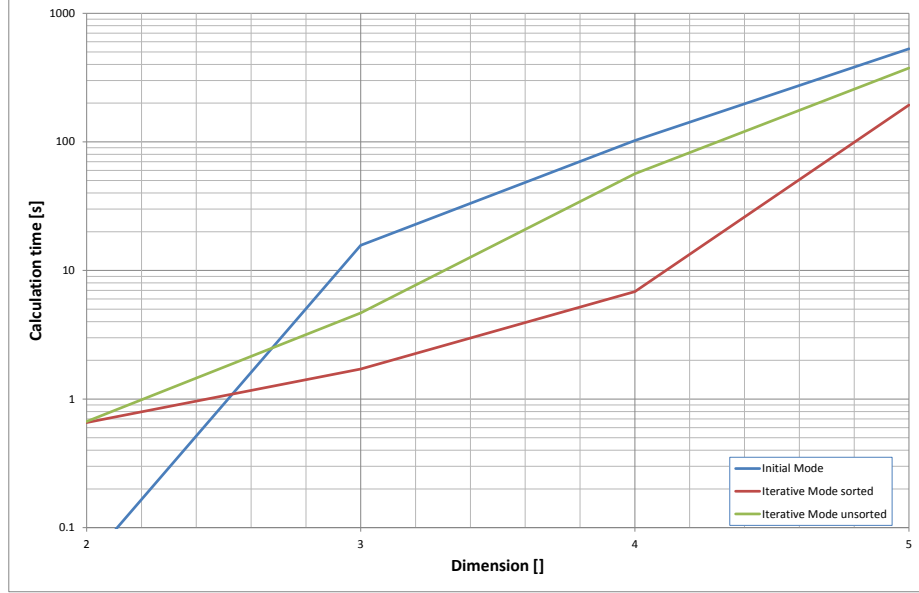
**Figure 72:** Run-time comparison between *RLR* and *Polymake*

#### 5.2.1.2 Iterative Mode of Recursive Light Ray Algorithm

The *RLR* algorithm supports an iterative append mode (see Subsection 5e). In this mode it is possible to modify an existing tessellation and append additional data points. The advantage of this mode is the reduced time effort to determine the Voronoi tessellation.

Figure 73 shows the comparison of calculation times for the initial mode (blue), the iterative mode for unsorted (green) and sorted (red) data points, which have to be added. The initial mode determines the nearest neighbor information at once of 5000 data points. The iterative mode starts with a tessellation, that includes 4950 data points. In the calcula-

tion time the algorithm adds 50 new data points to the tessellation. The method has been explained in Passage 5e.



**Figure 73:** Time to append 50 sorted and unsorted data points to the existing triangulation (4950 data points) with *RLR*

The blue graph shows the time effort in seconds, which is needed to calculate the whole tessellation at once. The calculation time is circa one magnitude higher than in the iterative mode. During a measurement process *AmmOC* determines the Voronoi tessellation, after new data points have been measured. The calculation time differs, if unsorted or sorted data points are added to the Tessellation. During a measurement process all data points are sorted, since *Dynamic Testdrive* defines measurement ramps, which are located in a very small sector of the data space. The time saving is caused by the smaller region, that has to be reconstructed by the iterative mode. The initial mode is faster than the iterative mode for dimensions below 3.

### 5.2.2 Stationary Solution

In this subsection, it is shown how the common measurement method *Full Factorial* (see Subsection 3.1.2) differs to the new *Dynamic Testdrive* approach. Therefore, we optimize the engine model provided by AVL. The optimization of this model includes the specific fuel consumption, critical temperature and pressure and emission values. For details see Subsection 4.4.3

The initial state of each operation point (see Equation (1)) in the solution is set to a specific fuel consumption value of  $1000 \frac{\text{g}}{\text{kWh}}$ . Each solution, with a better fuel consumption value, improves the quality of the optimization. Due to bad coverage, it is possible, that the algorithm does not find a valid solution point in a specific area. The quality of the

optimization is given by the sum of the specific fuel consumption values of all operation points. The results of the optimization are discussed in the following paragraphs.

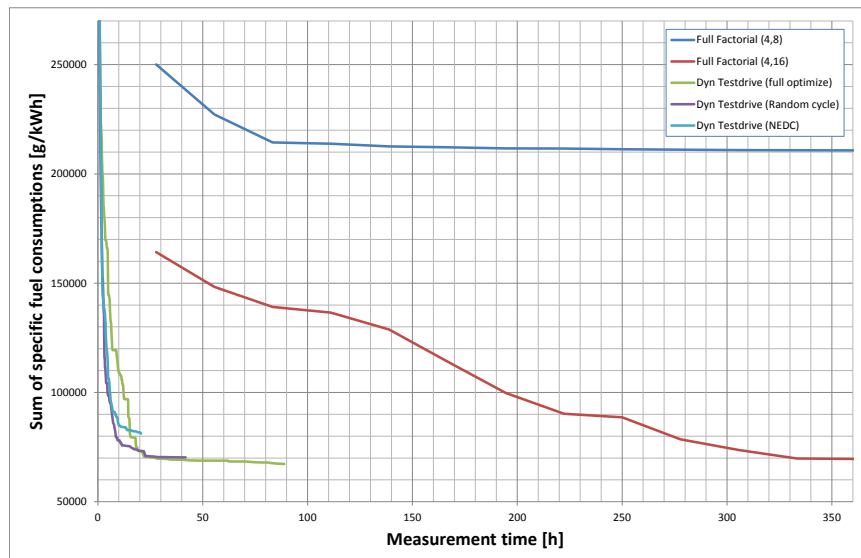
#### 5.2.2.1 Stationary Optimization of AVL engine model

The configuration of the system, which has to be analyzed is given in Subsection 7.2.2. The criteria of the optimization are the specific fuel consumption, the exhaust emissions of hydrocarbons, carbon-monoxide and nitrogen oxides.

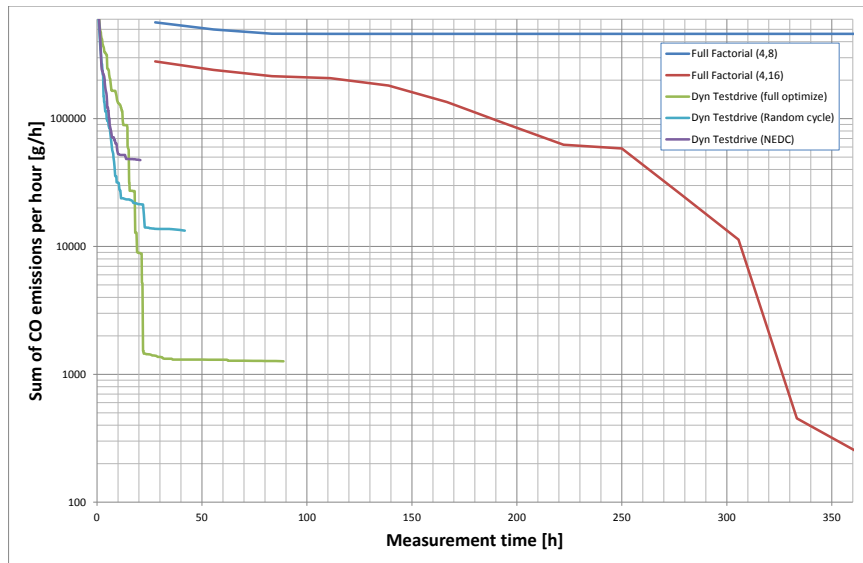
During this study, we perform full factorial variations (see Subsection 3.1.2) with different actuator resolutions. In order to improve the efficiency of the full factorial method, we increase the resolution of the dynamic actuators, which have a bigger impact on the objective function. In this case, we increase the resolution of *revolution frequency*, *injected fuel quantity* and *air filling*. A detailed description of all actuators is given in Subsection 2.2.1.2.

Besides the full factorial runs, the engine model has been optimized with the *Dynamic Test Drive* algorithm. The algorithm runs in different modes. The *full optimize* mode optimizes every criterion as good as possible, without taking any quantification cycle as reference. The other two operation modi focus on the real world driving cycle *RANDOM* and the established new emission driving cycle *NEDC* (see a description in Subsection 2.2.3).

Figure 74 shows the improvement of the sum of specific fuel consumption of the solution field during the measurement. The blue curve shows the worst development. The *Full Factorial* method resolves the important actuators with 8 divisions and all other actuators with 4 divisions. The main problem in this case is the poor resolution of the important actuators, that is why the solution field can not be occupied completely. The orange graph shows the same method, but with a higher resolution. This run is able to deliver a good stationary solution after 360 hours of measurement time.



**Figure 74:** Stationary solution quality of full factorial and *Dynamic Testdrive* runs. Sum of specific fuel consumption versus measurement time.

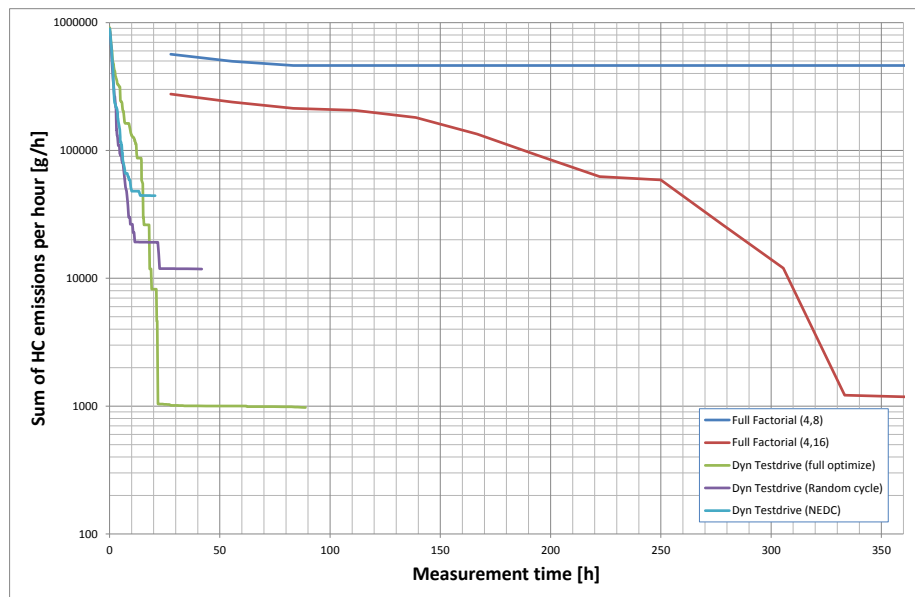


**Figure 75:** Stationary solution quality of full factorial and *Dynamic Testdrive* runs. Sum of carbon monoxide versus measurement time.

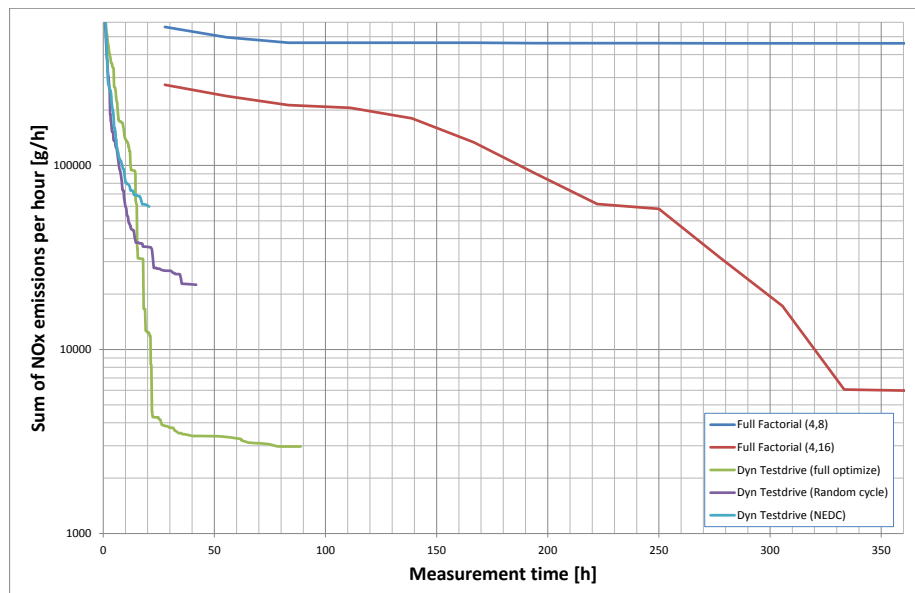
The three remaining graphs are created with the *Dynamic Testdrive* method. Each slope is much higher than the slope of the *Full Factorial* methods, so this method is able to deliver good stationary solutions in a very short time span. The green line reaches the smallest sum of specific fuel consumption, thus the best stationary solution. During this run the *Dynamic Testdrive* method tries to improve every criterion of the engine model. This run has been interrupted after 80 hours of measurement time.

The purple graph shows the improvement during the calibration of the engine model with respect to the *RANDOM* cycle. This process terminates, after all exhaust emission constraints have been fulfilled. The performance is quite similar to the green curve.

The last plot (light blue) shows the calibration in accordance with the *NEDC*. Just like the previous run, the method terminates after all constraints have been satisfied. The sum of specific fuel consumption is not as low as in the previous two cases. Furthermore, this run finishes after 20 hours of measurement. During this run, we focus on emission constraints, not on ideal fuel consumption. That is why the measurement time is that small. The figures 75, 76 and 77 show the development of the sum of exhaust emissions of carbon monoxides, hydro carbons and nitrogen oxides, accordingly. These figures show a similar behavior than the development of the sum of the specific fuel consumption. As before, the fuel factorial method (red line) is able to find as low results, than the *Dynamic Testdrive* method. Actually, the full factorial method found a lower sum of carbon oxides than the *Dynamic Testdrive* approach. Since the measurement runs with the *Dynamic Test Drive* method terminate with a higher level of carbon-monoxides, we may conclude, that it was not necessary for the fulfillment of the European exhaust emission regulations. The full factorial method did not terminate after it archived a certain exhaust emission level, but continue the measurement until all measurement points have been recorded.

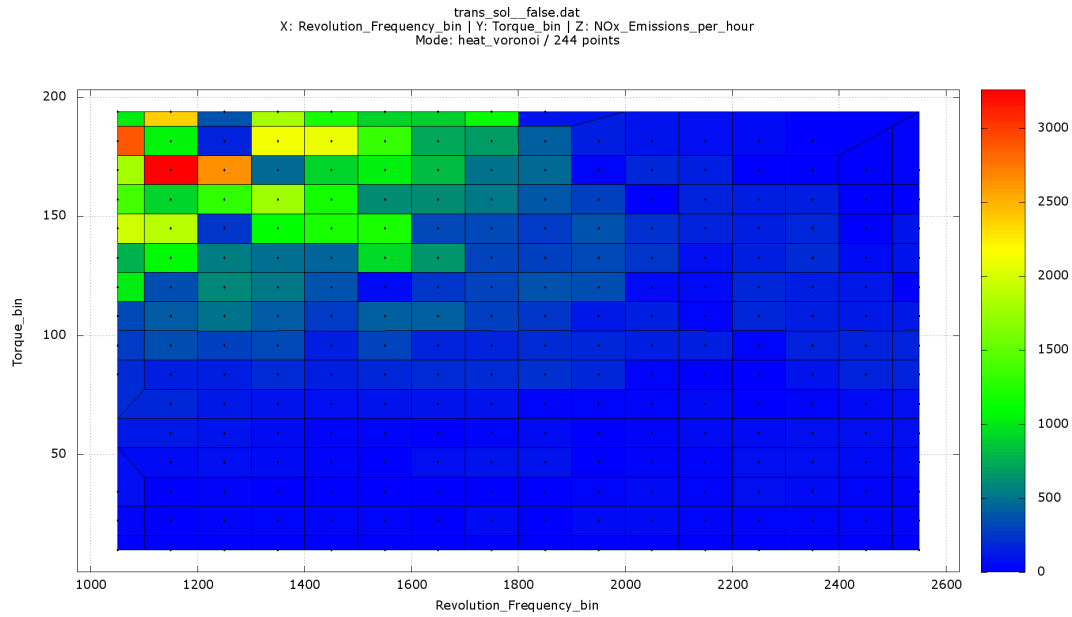


**Figure 76:** Stationary solution quality of full factorial and *Dynamic Testdrive* runs. Sum of hydro carbons versus measurement time.



**Figure 77:** Stationary solution quality of full factorial and *Dynamic Testdrive* runs. Sum of nitrogen oxides versus measurement time.

The optimization relating to the *RANDOM* cycle and *NEDC* shows, that the absolute level of emissions does not have to be minimal. The optimization with respect to *NEDC* terminated at a sum for CO of  $50000 \frac{g}{h}$ , HC of  $45000 \frac{g}{h}$  and NOx of  $60000 \frac{g}{h}$ . The reason for this lies within the European regulation. The quantification cycles do not cover all operation points, which are possible to drive with a car engine (see details in Subsection 2.2.3.1), therefore not all operation points are important for the optimization. An example is given in Figure 78. The *NEDC* covers operation points below 90 Nm of torque, only. We see clearly, that the emission values for nitrogen oxides jumps to very high values for operational points with an engine torque above 90 Nm. This aspect will be discussed in detail in the following Subsection 5.4.1.5.



**Figure 78:** Stationary solution for the *NEDC*. Nitrogen oxides emissions ( $g/h$ ) for different operation points.

---

## 5.3 Evaluation of preceding stages of AmmOC

---

In this passage, we compare the different stages of evolution of the project *AmmOC*. As summarized in Subsection 3.2, the project includes four different approaches. During the project lifetime of 8 man-years, the project team has implemented and tested the *Random Walker*, the *Deterministic Walker*, the *Greedy Ants* and the *Integer Linear Program* method.

This subsection illustrates the characteristics of each method. Furthermore, we compare all methods to the latest, i.e. the *Integer Linear Program* method. Finally, we do the qualitative evaluation of all methods.

---

### 5.3.1 Random Walker

---

#### 5.3.1.1 Description of Random Walker run

The optimization is done for a spark-ignited engine, with 1.2 liter displacement and two phase actuators for intake and exhaust valves. During the optimization, the project team vary the air-to-fuel ratio and the revolution frequency. The optimization criteria are the specific fuel consumption  $b_e$  in  $g/kWh$  and the critical exhaust temperature  $T_{EbTC1}$  in  $^{\circ}C$ .

The optimization is based on a small set of data points, which have been simulated with GT-Power (see [Mer10] for further information). The implementation of the *Random Walker* algorithm is explained in 3.2.1. The optimizer is configured, so that it covers 90 % of the total solution field (Revolution frequency vs. engine torque). The algorithm terminate after the creation of 1000 local optima.

#### 5.3.1.2 Summary of Results of Random Walker run

Figure 79 shows the data point distribution, which is the foundation for the optimization run. The measurement plan is created by an early version of the *Dynamic Testdrive* module. The result is comparable to Figure 49. The early method create measurement ramps with a constant measurement frequency. The start-/end-points are determined by an uniform random number generator. This method does not analyze the measurement density, in order to enforce regions, with a low measurement density. So, this algorithm create an overemphasis of the central data space region. Additionally, the data point acquisition does not measure all operation points, e.g.  $(2071 \text{ min}^{-1}, 15 \text{ Nm})$ . So, the optimization algorithm is not be able to create a complete local solution.

The *Random Walker* algorithm create 1000 solution with local optima. These solutions and the calculated rating of these solutions is summarized in Figure 80. The *Rating\_TOTAL* is given by a weighted combination of the sum, standard deviation and mean value of all score values of each operation point (see [Mer10] Equation 5.38).

Finally, the algorithm selects solution 168 as final transient solution. This solution is discussed in detail. The optimal settings of the phase angle of the exhaust camshaft is shown in Figure 81. Green fields show successfully chosen settings, red fields symbolize missing settings caused by missing data points or at least one excluding constraint. The grey fields are not defined by the *Random Walker*, since it has covered 90 % of the solution field, only.

The maximal gradient constraints for the phase angle of the exhaust valves were  $0.03 \frac{^{\circ}CA}{\text{min}^{-1}}$  and  $0.6 \frac{^{\circ}CA}{Nm}$ . We detect easily transitions between operation points, which violate these gradient constraints. So, the computed solution does not preserve all constraint, thus the solution has to be repaired.

---

<sup>40</sup> Source: taken from [Mer10] Figure 6.1

<sup>41</sup> Source: taken from [Mer10] Figure 6.1



[X]: ActDynoSpeed  
[Y]: ActDynoTorque

[Chess\_Coords] - [XY-Coords]  
[Events]

### Events

[0,7] - [2071.4,97.0] 0	[1,7] - [2280.5,97.0] 0	[2,7] - [2571.1,97.0] 0	[3,7] - [2849.2,97.0] 0	[4,7] - [3140.7,97.0] 0	[5,7] - [3419.9,97.0] 0	[6,7] - [3714.1,97.0] 0	[7,7] - [3922.1,97.0] 0
[0,6] - [2071.4,86.3] 16	[1,6] - [2280.5,86.3] 34	[2,6] - [2571.1,86.3] 31	[3,6] - [2849.2,86.3] 14	[4,6] - [3140.7,86.3] 18	[5,6] - [3419.9,86.3] 24	[6,6] - [3714.1,86.3] 30	[7,6] - [3922.1,86.3] 19
[0,5] - [2071.4,75.1] 36	[1,5] - [2280.5,75.1] 85	[2,5] - [2571.1,75.1] 75	[3,5] - [2849.2,75.1] 57	[4,5] - [3140.7,75.1] 63	[5,5] - [3419.9,75.1] 54	[6,5] - [3714.1,75.1] 51	[7,5] - [3922.1,75.1] 17
[0,4] - [2071.4,63.0] 48	[1,4] - [2280.5,63.0] 117	[2,4] - [2571.1,63.0] 105	[3,4] - [2849.2,63.0] 81	[4,4] - [3140.7,63.0] 70	[5,4] - [3419.9,63.0] 67	[6,4] - [3714.1,63.0] 69	[7,4] - [3922.1,63.0] 37
[0,3] - [2071.4,50.3] 31	[1,3] - [2280.5,50.3] 44	[2,3] - [2571.1,50.3] 67	[3,3] - [2849.2,50.3] 76	[4,3] - [3140.7,50.3] 71	[5,3] - [3419.9,50.3] 66	[6,3] - [3714.1,50.3] 56	[7,3] - [3922.1,50.3] 29
[0,2] - [2071.4,38.1] 15	[1,2] - [2280.5,38.1] 24	[2,2] - [2571.1,38.1] 35	[3,2] - [2849.2,38.1] 40	[4,2] - [3140.7,38.1] 48	[5,2] - [3419.9,38.1] 54	[6,2] - [3714.1,38.1] 64	[7,2] - [3922.1,38.1] 23
[0,1] - [2071.4,26.0] 0	[1,1] - [2280.5,26.0] 10	[2,1] - [2571.1,26.0] 11	[3,1] - [2849.2,26.0] 27	[4,1] - [3140.7,26.0] 30	[5,1] - [3419.9,26.0] 40	[6,1] - [3714.1,26.0] 27	[7,1] - [3922.1,26.0] 3
[0,0] - [2071.4,15.0] 0	[1,0] - [2280.5,15.0] 0	[2,0] - [2571.1,15.0] 3	[3,0] - [2849.2,15.0] 0	[4,0] - [3140.7,15.0] 18	[5,0] - [3419.9,15.0] 13	[6,0] - [3714.1,15.0] 7	[7,0] - [3922.1,15.0] 0

Figure 79: Data point distribution during *Random Walker* run<sup>40</sup>

Rank	Run	SUM	SIGMA	AVG	Rating_TOTAL
1	RW1_RUN_168	4.890049	0.065792	0.106305	3.635878
2	RW1_RUN_114	4.925151	0.083381	0.111935	3.523896
3	RW2_RUN_362	4.808258	0.075233	0.106850	3.519246
4	RW1_RUN_355	4.887792	0.082887	0.111086	3.508138

Figure 80: Summary of local optima found with *Random Walker* algorithm<sup>41</sup>

### CAM\_ex

[0,7] - [2071.4,97.0] XXX	[1,7] - [2280.5,97.0] n.v.	[2,7] - [2571.1,97.0] n.v.	[3,7] - [2849.2,97.0] n.v.	[4,7] - [3140.7,97.0] n.v.	[5,7] - [3419.9,97.0] XXX	[6,7] - [3714.1,97.0] XXX	[7,7] - [3922.1,97.0] n.v.
[0,6] - [2071.4,86.3] GRAD!	[1,6] - [2280.5,86.3] 246.1 +/- 1.7834	[2,6] - [2571.1,86.3] 246.1 +/- 1.7834	[3,6] - [2849.2,86.3] 258.1 +/- 1.6561	[4,6] - [3140.7,86.3] 276.1 +/- 1.8031	[5,6] - [3419.9,86.3] 276.1 +/- 1.8031	[6,6] - [3714.1,86.3] 258.1 +/- 1.6561	[7,6] - [3922.1,86.3] n.v.
[0,5] - [2071.4,75.1] 240.0 +/- 1.8201	[1,5] - [2280.5,75.1] 246.1 +/- 1.7834	[2,5] - [2571.1,75.1] 263.9 +/- 1.7143	[3,5] - [2849.2,75.1] 258.1 +/- 1.6561	[4,5] - [3140.7,75.1] 276.1 +/- 1.8031	[5,5] - [3419.9,75.1] 276.1 +/- 1.8031	[6,5] - [3714.1,75.1] 246.1 +/- 1.7834	[7,5] - [3922.1,75.1] 282.0 +/- 1.6504
[0,4] - [2071.4,63.0] 240.0 +/- 1.8201	[1,4] - [2280.5,63.0] 282.0 +/- 1.6504	[2,4] - [2571.1,63.0] 282.0 +/- 1.6504	[3,4] - [2849.2,63.0] 276.1 +/- 1.8031	[4,4] - [3140.7,63.0] 251.7 +/- 1.6949	[5,4] - [3419.9,63.0] 282.0 +/- 1.6504	[6,4] - [3714.1,63.0] 240.0 +/- 1.8201	[7,4] - [3922.1,63.0] 276.1 +/- 1.8031
[0,3] - [2071.4,50.3] 287.5 +/- 1.4583	[1,3] - [2280.5,50.3] 276.1 +/- 1.8031	[2,3] - [2571.1,50.3] 270.2 +/- 1.6877	[3,3] - [2849.2,50.3] 270.2 +/- 1.6877	[4,3] - [3140.7,50.3] 282.0 +/- 1.6504	[5,3] - [3419.9,50.3] 246.1 +/- 1.7834	[6,3] - [3714.1,50.3] 251.7 +/- 1.6949	[7,3] - [3922.1,50.3] 246.1 +/- 1.7834
[0,2] - [2071.4,38.1] 230.5 +/- 0.3059	[1,2] - [2280.5,38.1] 263.9 +/- 1.7143	[2,2] - [2571.1,38.1] 270.2 +/- 1.6877	[3,2] - [2849.2,38.1] 276.1 +/- 1.8031	[4,2] - [3140.7,38.1] 240.0 +/- 1.8201	[5,2] - [3419.9,38.1] 246.1 +/- 1.7834	[6,2] - [3714.1,38.1] 246.1 +/- 1.7834	[7,2] - [3922.1,38.1] GRAD!
[0,1] - [2071.4,26.0] XXX	[1,1] - [2280.5,26.0] 282.0 +/- 1.6504	[2,1] - [2571.1,26.0] 282.0 +/- 1.6504	[3,1] - [2849.2,26.0] 287.5 +/- 1.4583	[4,1] - [3140.7,26.0] 287.5 +/- 1.4583	[5,1] - [3419.9,26.0] 282.0 +/- 1.6504	[6,1] - [3714.1,26.0] 240.0 +/- 1.8201	[7,1] - [3922.1,26.0] GRAD!
[0,0] - [2071.4,15.0] XXX	[1,0] - [2280.5,15.0] XXX	[2,0] - [2571.1,15.0] GRAD!	[3,0] - [2849.2,15.0] XXX	[4,0] - [3140.7,15.0] 276.1 +/- 1.8031	[5,0] - [3419.9,15.0] 282.0 +/- 1.6504	[6,0] - [3714.1,15.0] 246.1 +/- 1.7834	[7,0] - [3922.1,15.0] XXX

Figure 81: Optimal settings for phase of exhaust valve derived with *Random Walker* algorithm<sup>42</sup>

The objective function of this optimizer is a combination of the specific fuel consumption  $b_e$  and the critical exhaust gas temperature. The temperature has to be below a threshold value in order not to damage the engine itself. The resulting specific fuel consumption of solution 168 is given in Figure 82, the resulting temperature is shown in Figure 83. Figure 84 shows the evaluation of Equation 5.38 in [Mer10]. The solution shows, that the optimizer has taken data points with a positive total rating, but it is forced to include one single data point with a negative rating. The negative rating is caused by a high exhaust gas temperature. 1106.8°C is above the defined upper threshold.

**b<sub>e</sub>**

[0,7] - [2071.4,97.0] <b>XXX</b>	[1,7] - [2280.5,97.0] n.v.	[2,7] - [2571.1,97.0] n.v.	[3,7] - [2849.2,97.0] n.v.	[4,7] - [3140.7,97.0] n.v.	[5,7] - [3419.9,97.0] <b>XXX</b>	[6,7] - [3714.1,97.0] <b>XXX</b>	[7,7] - [3922.1,97.0] n.v.
[0,6] - [2071.4,86.3] <b>GRAD!</b>	[1,6] - [2280.5,86.3] <b>254.0</b> +/- 0.0	[2,6] - [2571.1,86.3] <b>247.0</b> +/- 0.0	[3,6] - [2849.2,86.3] <b>276.0</b> +/- 0.0	[4,6] - [3140.7,86.3] <b>250.0</b> +/- 0.0	[5,6] - [3419.9,86.3] <b>255.0</b> +/- 0.0	[6,6] - [3714.1,86.3] <b>281.0</b> +/- 0.0	[7,6] - [3922.1,86.3] n.v.
[0,5] - [2071.4,75.1] <b>260.0</b> +/- 0.0	[1,5] - [2280.5,75.1] <b>249.0</b> +/- 0.0	[2,5] - [2571.1,75.1] <b>271.0</b> +/- 0.0	[3,5] - [2849.2,75.1] <b>258.0</b> +/- 0.0	[4,5] - [3140.7,75.1] <b>317.0</b> +/- 0.0	[5,5] - [3419.9,75.1] <b>258.0</b> +/- 0.0	[6,5] - [3714.1,75.1] <b>291.0</b> +/- 0.0	[7,5] - [3922.1,75.1] <b>310.0</b> +/- 0.0
[0,4] - [2071.4,63.0] <b>250.0</b> +/- 0.0	[1,4] - [2280.5,63.0] <b>266.5</b> +/- 0.5000	[2,4] - [2571.1,63.0] <b>263.0</b> +/- 0.0	[3,4] - [2849.2,63.0] <b>260.0</b> +/- 0.0	[4,4] - [3140.7,63.0] <b>277.0</b> +/- 0.0	[5,4] - [3419.9,63.0] <b>276.0</b> +/- 0.0	[6,4] - [3714.1,63.0] <b>254.0</b> +/- 0.0	[7,4] - [3922.1,63.0] <b>289.0</b> +/- 0.0
[0,3] - [2071.4,50.3] <b>262.0</b> +/- 0.0	[1,3] - [2280.5,50.3] <b>273.0</b> +/- 0.0	[2,3] - [2571.1,50.3] <b>268.0</b> +/- 0.0	[3,3] - [2849.2,50.3] <b>282.0</b> +/- 0.0	[4,3] - [3140.7,50.3] <b>265.0</b> +/- 0.0	[5,3] - [3419.9,50.3] <b>300.0</b> +/- 0.0	[6,3] - [3714.1,50.3] <b>321.0</b> +/- 0.0	[7,3] - [3922.1,50.3] <b>269.0</b> +/- 0.0
[0,2] - [2071.4,38.1] <b>296.0</b> +/- 0.0	[1,2] - [2280.5,38.1] <b>315.0</b> +/- 0.0	[2,2] - [2571.1,38.1] <b>322.0</b> +/- 0.0	[3,2] - [2849.2,38.1] <b>358.0</b> +/- 0.0	[4,2] - [3140.7,38.1] <b>329.0</b> +/- 0.0	[5,2] - [3419.9,38.1] <b>359.0</b> +/- 0.0	[6,2] - [3714.1,38.1] <b>348.0</b> +/- 0.0	[7,2] - [3922.1,38.1] <b>GRAD!</b>
[0,1] - [2071.4,26.0] <b>XXX</b>	[1,1] - [2280.5,26.0] <b>480.0</b> +/- 0.0	[2,1] - [2571.1,26.0] <b>595.0</b> +/- 0.0	[3,1] - [2849.2,26.0] <b>366.0</b> +/- 0.0	[4,1] - [3140.7,26.0] <b>386.0</b> +/- 0.0	[5,1] - [3419.9,26.0] <b>433.0</b> +/- 0.0	[6,1] - [3714.1,26.0] <b>521.0</b> +/- 0.0	[7,1] - [3922.1,26.0] <b>GRAD!</b>
[0,0] - [2071.4,15.0] <b>XXX</b>	[1,0] - [2280.5,15.0] <b>XXX</b>	[2,0] - [2571.1,15.0] <b>GRAD!</b>	[3,0] - [2849.2,15.0] <b>XXX</b>	[4,0] - [3140.7,15.0] <b>745.0</b> +/- 0.0	[5,0] - [3419.9,15.0] <b>716.0</b> +/- 0.0	[6,0] - [3714.1,15.0] <b>621.0</b> +/- 0.0	[7,0] - [3922.1,15.0] <b>XXX</b>

**Figure 82:** Resulting optimal specific fuel consumption derived with *Random Walker* algorithm<sup>43</sup>

**T<sub>EbTC1</sub>**

[0,7] - [2071.4,97.0] <b>XXX</b>	[1,7] - [2280.5,97.0] n.v.	[2,7] - [2571.1,97.0] n.v.	[3,7] - [2849.2,97.0] n.v.	[4,7] - [3140.7,97.0] n.v.	[5,7] - [3419.9,97.0] <b>XXX</b>	[6,7] - [3714.1,97.0] <b>XXX</b>	[7,7] - [3922.1,97.0] n.v.
[0,6] - [2071.4,86.3] <b>GRAD!</b>	[1,6] - [2280.5,86.3] <b>866.9</b> +/- 0.0	[2,6] - [2571.1,86.3] <b>896.9</b> +/- 0.0	[3,6] - [2849.2,86.3] <b>856.9</b> +/- 0.0	[4,6] - [3140.7,86.3] <b>946.9</b> +/- 0.0	[5,6] - [3419.9,86.3] <b>1026.8</b> +/- 0.0	[6,6] - [3714.1,86.3] <b>876.9</b> +/- 0.0	[7,6] - [3922.1,86.3] n.v.
[0,5] - [2071.4,75.1] <b>896.9</b> +/- 0.0	[1,5] - [2280.5,75.1] <b>806.9</b> +/- 0.0	[2,5] - [2571.1,75.1] <b>866.9</b> +/- 0.0	[3,5] - [2849.2,75.1] <b>966.9</b> +/- 0.0	[4,5] - [3140.7,75.1] <b>826.9</b> +/- 0.0	[5,5] - [3419.9,75.1] <b>896.9</b> +/- 0.0	[6,5] - [3714.1,75.1] <b>846.9</b> +/- 0.0	[7,5] - [3922.1,75.1] <b>836.9</b> +/- 0.0
[0,4] - [2071.4,63.0] <b>796.9</b> +/- 0.0	[1,4] - [2280.5,63.0] <b>871.9</b> +/- 35.0000	[2,4] - [2571.1,63.0] <b>866.9</b> +/- 0.0	[3,4] - [2849.2,63.0] <b>906.9</b> +/- 0.0	[4,4] - [3140.7,63.0] <b>866.9</b> +/- 0.0	[5,4] - [3419.9,63.0] <b>1006.9</b> +/- 0.0	[6,4] - [3714.1,63.0] <b>956.9</b> +/- 0.0	[7,4] - [3922.1,63.0] <b>1086.8</b> +/- 0.0
[0,3] - [2071.4,50.3] <b>746.9</b> +/- 0.0	[1,3] - [2280.5,50.3] <b>846.9</b> +/- 0.0	[2,3] - [2571.1,50.3] <b>876.9</b> +/- 0.0	[3,3] - [2849.2,50.3] <b>916.9</b> +/- 0.0	[4,3] - [3140.7,50.3] <b>886.9</b> +/- 0.0	[5,3] - [3419.9,50.3] <b>976.9</b> +/- 0.0	[6,3] - [3714.1,50.3] <b>956.9</b> +/- 0.0	[7,3] - [3922.1,50.3] <b>946.9</b> +/- 0.0
[0,2] - [2071.4,38.1] <b>896.9</b> +/- 0.0	[1,2] - [2280.5,38.1] <b>986.9</b> +/- 0.0	[2,2] - [2571.1,38.1] <b>966.9</b> +/- 0.0	[3,2] - [2849.2,38.1] <b>946.9</b> +/- 0.0	[4,2] - [3140.7,38.1] <b>956.9</b> +/- 0.0	[5,2] - [3419.9,38.1] <b>1006.9</b> +/- 0.0	[6,2] - [3714.1,38.1] <b>1066.8</b> +/- 0.0	[7,2] - [3922.1,38.1] <b>GRAD!</b>
[0,1] - [2071.4,26.0] <b>XXX</b>	[1,1] - [2280.5,26.0] <b>1026.8</b> +/- 0.0	[2,1] - [2571.1,26.0] <b>1106.8</b> +/- 0.0	[3,1] - [2849.2,26.0] <b>906.9</b> +/- 0.0	[4,1] - [3140.7,26.0] <b>1056.8</b> +/- 0.0	[5,1] - [3419.9,26.0] <b>1076.8</b> +/- 0.0	[6,1] - [3714.1,26.0] <b>1066.8</b> +/- 0.0	[7,1] - [3922.1,26.0] <b>GRAD!</b>
[0,0] - [2071.4,15.0] <b>XXX</b>	[1,0] - [2280.5,15.0] <b>XXX</b>	[2,0] - [2571.1,15.0] <b>GRAD!</b>	[3,0] - [2849.2,15.0] <b>XXX</b>	[4,0] - [3140.7,15.0] <b>1046.8</b> +/- 0.0	[5,0] - [3419.9,15.0] <b>1006.9</b> +/- 0.0	[6,0] - [3714.1,15.0] <b>986.9</b> +/- 0.0	[7,0] - [3922.1,15.0] <b>XXX</b>

**Figure 83:** Resulting optimal exhaust gas temperature derived with *Random Walker* algorithm<sup>44</sup>

<sup>42</sup> Source: taken from [Mer10] Figure 6.4

<sup>43</sup> Source: taken from [Mer10] Figure 6.7

<sup>44</sup> Source: taken from [Mer10] Figure 6.5

<sup>45</sup> Source: taken from [Mer10] Figure 6.8

## Rating

[0,7] - [2071.4,97.0] XXX	[1,7] - [2280.5,97.0] n.v.	[2,7] - [2571.1,97.0] n.v.	[3,7] - [2849.2,97.0] n.v.	[4,7] - [3140.7,97.0] n.v.	[5,7] - [3419.9,97.0] XXX	[6,7] - [3714.1,97.0] XXX	[7,7] - [3922.1,97.0] n.v.
[0,6] - [2071.4,86.3] GRAD!	[1,6] - [2280.5,86.3] 0.044255	[2,6] - [2571.1,86.3] 0.005677	[3,6] - [2849.2,86.3] 0.064185	[4,6] - [3140.7,86.3] 0.014728	[5,6] - [3419.9,86.3] 0.034801	[6,6] - [3714.1,86.3] 0.046886	[7,6] - [3922.1,86.3] n.v.
[0,5] - [2071.4,75.1] 0.030168	[1,5] - [2280.5,75.1] 0.039987	[2,5] - [2571.1,75.1] 0.050086	[3,5] - [2849.2,75.1] 0.047243	[4,5] - [3140.7,75.1] 0.049694	[5,5] - [3419.9,75.1] 0.047768	[6,5] - [3714.1,75.1] 0.088589	[7,5] - [3922.1,75.1] 0.110506
[0,4] - [2071.4,63.0] 0.116130	[1,4] - [2280.5,63.0] 0.079507	[2,4] - [2571.1,63.0] 0.116668	[3,4] - [2849.2,63.0] 0.101150	[4,4] - [3140.7,63.0] 0.081754	[5,4] - [3419.9,63.0] 0.092093	[6,4] - [3714.1,63.0] 0.104996	[7,4] - [3922.1,63.0] 0.050704
[0,3] - [2071.4,50.3] 0.155658	[1,3] - [2280.5,50.3] 0.155437	[2,3] - [2571.1,50.3] 0.146768	[3,3] - [2849.2,50.3] 0.102907	[4,3] - [3140.7,50.3] 0.152369	[5,3] - [3419.9,50.3] 0.123951	[6,3] - [3714.1,50.3] 0.037135	[7,3] - [3922.1,50.3] 0.219787
[0,2] - [2071.4,38.1] 0.246315	[1,2] - [2280.5,38.1] 0.174750	[2,2] - [2571.1,38.1] 0.133692	[3,2] - [2849.2,38.1] 0.132067	[4,2] - [3140.7,38.1] 0.171495	[5,2] - [3419.9,38.1] 0.085030	[6,2] - [3714.1,38.1] 0.160174	[7,2] - [3922.1,38.1] GRAD!
[0,1] - [2071.4,26.0] XXX	[1,1] - [2280.5,26.0] 0.120422	[2,1] - [2571.1,26.0] -0.046862	[3,1] - [2849.2,26.0] 0.276672	[4,1] - [3140.7,26.0] 0.190170	[5,1] - [3419.9,26.0] 0.169331	[6,1] - [3714.1,26.0] 0.148098	[7,1] - [3922.1,26.0] GRAD!
[0,0] - [2071.4,15.0] XXX	[1,0] - [2280.5,15.0] XXX	[2,0] - [2571.1,15.0] GRAD!	[3,0] - [2849.2,15.0] XXX	[4,0] - [3140.7,15.0] 0.077133	[5,0] - [3419.9,15.0] 0.128849	[6,0] - [3714.1,15.0] 0.211123	[7,0] - [3922.1,15.0] XXX

**Figure 84:** Resulting optimal rating derived with *Random Walker* algorithm<sup>45</sup>

### 5.3.1.3 Conclusion of the Random Walker run

The previous paragraph summarize the results of M. Mertz et al. [Mer10]. Now, we are able to determine the differences and the similarities of the *Random Walker* approach in comparison to the approach of this work.

The *Random Walker* algorithm applies the hard boundary constraint, by adding a penalty function to the objective function. This is shown for the critical temperature in Figure 83. The *Integer Linear Program* maintains the critical value constraints (see Equation (121)) by erasing these data points and interrupting the current measurement ramps. That is why, it is possible, in the *Random Walker* approach, to add data points to the solution, which violate the critical value constraint.

Moreover, the *Random Walker* approach guarantees the transition constraints (see Equation (8)) on the taken path, only. This is shown in the analysis of the ideal phase angle settings of the exhaust camshaft in Figure 81. The implementation of this work maintains all tolerance constraints.

One more difference is the incompleteness of the derived local solutions. Even if the measurement process, is able to cover all operation points, the *Random Walker* algorithm creates samples, i.e. incomplete local solution, only. This limitation is necessary, since the gradient constraint, does not have to be fulfilled, if the *Random Walker* passes an operation point that has been assigned by the same walker previously. So, the fraction of broken constraints depends strongly on the length of the path.

Finally, the optimizer derives a large number of local samples, i.e. local partial solutions. In order to determine the global optimal solution, the algorithm compares all local optima and defines the best found as global solution. So, the algorithm is unable to proof the correctness of the global solution. In the *Integer Liner Program* approach the global solution is derived with the Branch and Bound/Cut method. This exact method is able to proof the correctness of the chosen global optimum.

The *Random Walker* implementation is the first stage of the *AmmOC* project. Many basic definitions remain till the current version of the project. The very basic idea of a discrete optimization algorithm is still valid. The discrete data points are classified by an arbitrary histogram. The resolution of this histogram is the resolution of the possible start and end point distribution. Figure 79 shows the data point distribution of the *Random Walker* run.

---

This illustration shows the identical information than Figure 50. So, the module *Dynamic testdrive* has been updated with weighted random number and Dijkstra routing ability. The basic idea is still valid. Additionally, the definition of data space and solution space is still equivalent. The data space (see Subsection 4.2.1) is given by all dynamic actuators, the solution space is given by actuators and result values, which are assigned as solution field, e.g. revolution frequency and engine torque.

The optimization include constraints in order to derive partial transient solutions. These solutions are optimized with more than one criterion, in this example, the solution is optimal in specific fuel consumption, running smoothness and critical temperatures. The basic idea of the optimization is very different from the current implementation, but an optimization, which includes exhaust emissions is theoretically possible.



---

### 5.3.2 Deterministic Walker

---

#### 5.3.2.1 Description of Deterministic Walker run

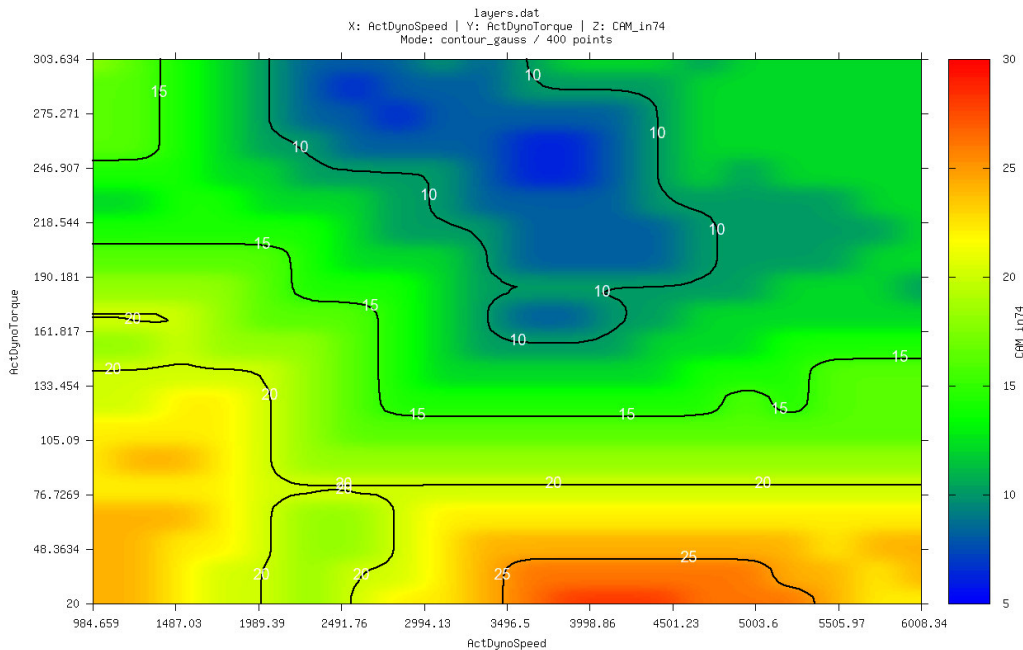
This optimization is done for a turbo-charged spark-ignited engine, with 1.6 liter displacement and two phase actuators for intake and exhaust vales and direct injection. The dynamic actuators are revolution frequency, engine torque, phase angle of exhaust and intake camshaft. The engine test bench is revolution frequency and engine torque controlled, so the revolution frequency and the engine torque are treated like actuators. The control software choose the spark-timing, opening angle of the air valve and break current in order to maintain the requested revolution frequency and engine torque. The observable specific fuel consumption has been measured and is the foundation of the objective function. All data points are real measurement data. The number of data points is 1.1 million after filter processes. The implementation of the *Deterministic Walker* algorithm is explained in Subsection 3.2.2.

#### 5.3.2.2 Summary of Results of Deterministic Walker run

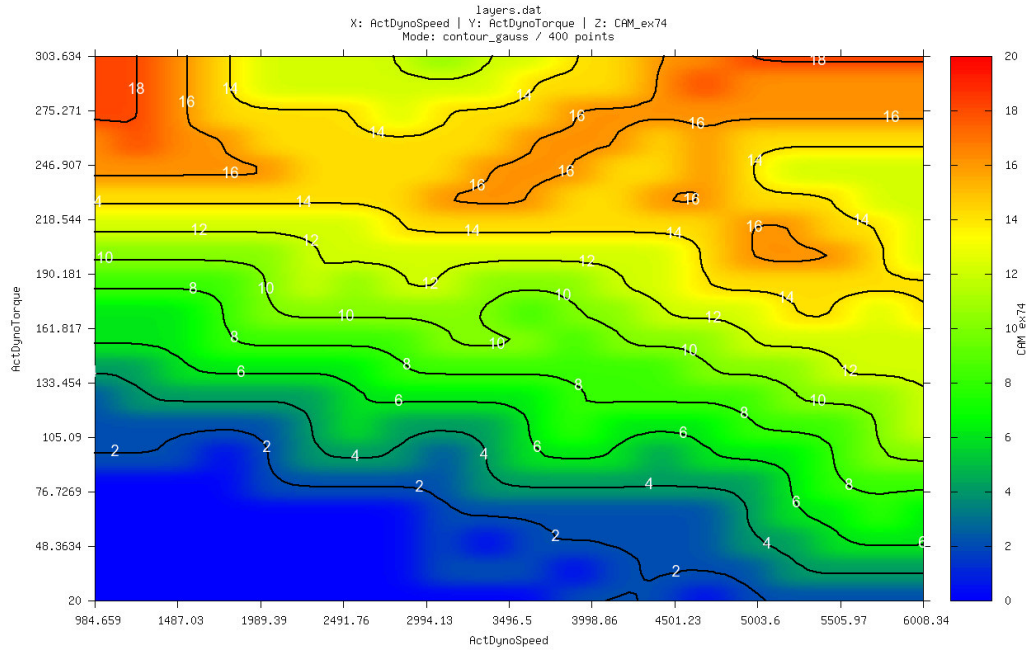
The data has been measured on an analytic spiral measurement path. Within a spiral run, the test bench varied the phase angles of exhaust and intake valve. This kind of measurement has been repeated for different combinations of revolution frequency and engine torque.

First, the algorithm created a model, which is based on the measured data. Afterwards, the *Deterministic Walker* constructs the local solutions with the random surface and the seed surface approach (for further information see [Geb11] Section 4.3).

The resulting optimal settings for the phase angle of the intake and exhaust camshaft is shown in Figure 85 and 86. One notices clearly, that the solution is transient, so the tolerance constraints (see Equation (8)) are completely fulfilled. The maximal gradient is equal to the one in the *Random Walker* run.



**Figure 85:** Optimal settings for phase of intake valve derived with *Deterministic Walker* algorithm<sup>46</sup>



**Figure 86:** Optimal settings for phase of exhaust valve derived with *Deterministic Walker* algorithm<sup>47</sup>

Figure 87 illustrates the objective function of the *Deterministic Walker* optimizer. As mentioned before, the objective function is based on the specific fuel consumption, only. So, the optimization routine tried to pick the best specific fuel consumption values of the model in each operation point. One notices, that in the right lower corner, the optimizer is forced to pick data points with non-ideal fuel consumption values.

### 5.3.2.3 Conclusion of *Deterministic Walker* run

The previous paragraph summarize the results of M. Gebhard et al. [Geb11]. Now, we are able to determine the differences and the similarities of the *Deterministic Walker* approach in comparison to the one of this work.

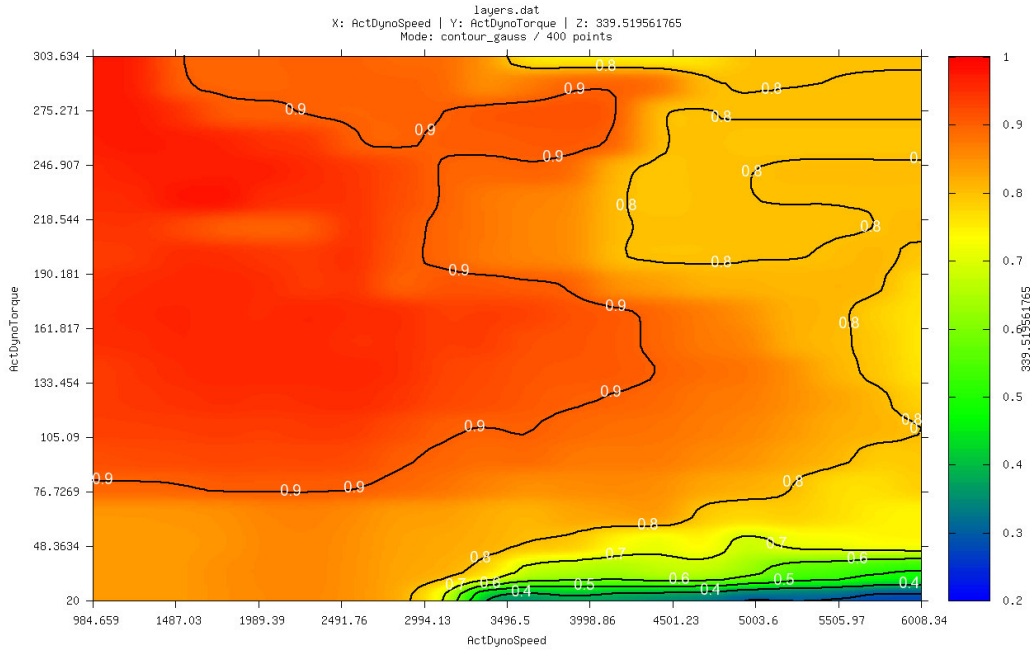
The *Deterministic Walker* treats hard boundaries in the same way than the *Random Walker* algorithm, so it is possible to add data points to the solution, which does not preserve the hard boundary constraints. The biggest difference to the previous approach and the *Integer Linear Program* approach is the model-based optimization. This approach forms an optimization model with the support vector model *SVM* approach or the complete rendering process of the data space with a distance weighted average function (see Subsection 3.2.2). So, the local optima do not include real measurement data, i.e. interpreted data points of the model. As in the previous approach, this method derives its global optima by comparing local optima. The algorithm is unable to proof the correctness of the assigned global optimum. This is possible in the current approach.

The *Deterministic Walker* approach is the second stage of the *AmmOC* project and a direct expansion of the *Random Walker* idea. The main purpose of the *Deterministic Walker* algo-

<sup>46</sup> Source: taken from [Geb11] Figure 5.4

<sup>47</sup> Source: taken from [Geb11] Figure 5.2

<sup>48</sup> Source: taken from [Geb11] Figure 5.6



**Figure 87:** Resulting optimal rating derived with *Deterministic Walker* algorithm<sup>48</sup>

rithm is the creation of complete local optima, which maintain all tolerance constraints. This ability is preserved in the current approach, too.

Due to the longer run-time compared to the *Random Walker* algorithm, this approach has to choose its seed points by statistical decision based on the local samples of the *Random Walkers*. So, this approach is able to improve its run-time by combination of both algorithms. The definition of data space and solution space were not modified, therefore they are still valid. Also, the formulation of the objective function is equal to the *Random Walker* approach. This definition differs to the current implementation, due to the same reasons, which are discussed in the previous section.

---

### 5.3.3 Greedy Ants

---

#### 5.3.3.1 Description of Greedy Ants run

The optimization has been done for the *Virtual Test Bench* engine model (see Subsection 4.2.6). In the work of P. Lind et al. [Lin12] the project team vary the revolution frequency and spark-timing. In a second calculation, they vary the angle of the air valve, too. So, they are able to generate different operation points (see Equation (1)).

The optimization criteria are the specific fuel consumption, critical inner cylinder pressure and critical exhaust gas temperature. The implementation of the evolutionary program is explained in Subsection 3.2.3. The optimization has been done online, i.e. during the measurement, so the algorithm is able to influence the creation of measurement plans. The first optimization has to derive a global optimum of a two dimensional problem. The second run derive the global optimum of a three dimensional problem. These optimization runs are explained in the following passage.

#### 5.3.3.2 Summary of Results of Greedy Ants run

First, the algorithm has to transform the discrete data points into a undirected graph (see Subsection 3.2.3). This combination of the nearest neighbor and the tolerance information is basis of the following optimization. The graph is shown in Figure 88, x-axis is given by normalized revolution frequency, y-axis is given by normalized spark timing. This benchmark problem shows a very simple systematical behavior. During this measurement the air valve, which defines mainly the engine torque, has been opened completely. The lambda value is set to 1, so the injected fuel quantity is constant for constant revolution frequency. That is why, the spark-timing shows a very simple behavior, if the spark-angle is optimal, the engine will produce a maximal amount of torque by consuming a constant amount of fuel. This is illustrated in Figure 88, the optimal data points, which are given by blue and red dots, are located in a small band.

Figure 89 shows the genetic imprinting of several *Chief* and *Scout* Ants (see Subsection 3.2.3). So, the algorithm is able to derive a global optimum. This is done by the *Quantifier Ant* (see Subsection 3.2.3). In General, this kind of ant constructs the local solution by following the strongest imprinting. Finally, the *Queen Ant* compares all *Quantifier Ants* and modifies the parameters of recombination and reproduction of all ants. This global optimum is shown in Figure 90. This optimum maintains the hard boundaries and the tolerance constraints.

This approach is applicable for problem with a larger number of dynamic actuators. Due to long run-times, the second evaluation focuses on a three dimensional problem. This run determines the global optimum of the spark-timing for different operation points. The results are shown in Figure 91. This solution fulfills the critical constraint (see Equation (5)) and the tolerance constraints (see Equation (8)).

The application of this algorithm on more optimization criteria is possible by the modification of the objective function. This is discussed in [Lin12] Equation 4.22. In principle, the optimization focuses on the specific fuel consumption, first. After the first determination of the global optimum, the algorithm evaluates the additional constraints, e.g. maximal integral emissions (see Equation (6)). If any constraint is violated, the algorithm re-weights

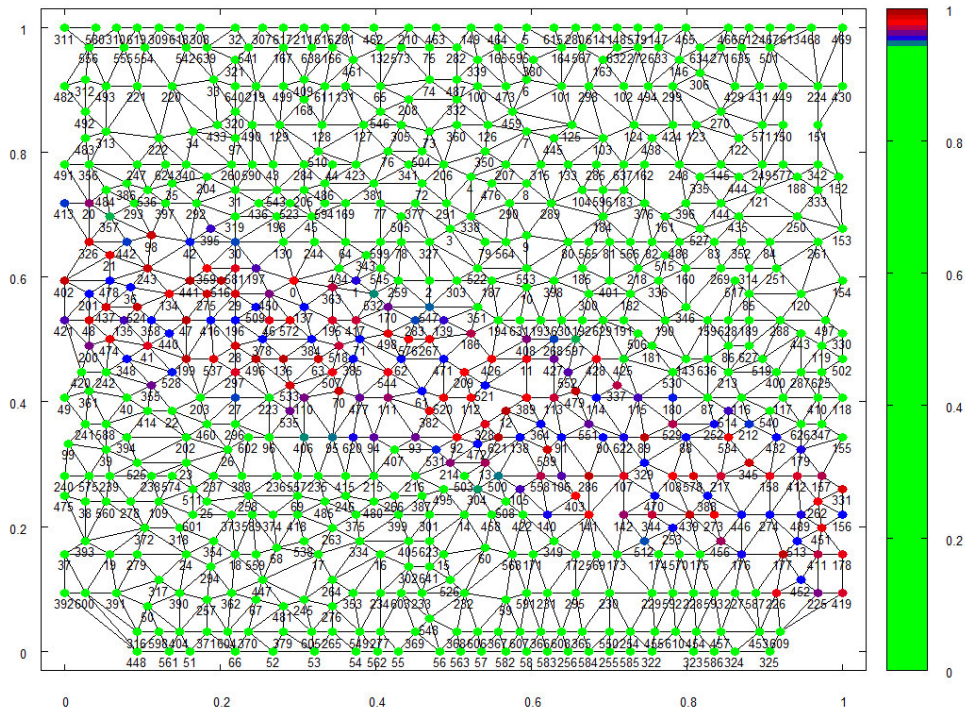
---

<sup>49</sup> Source: taken from [Lin12] Figure 53

<sup>50</sup> Source: taken from [Lin12] Figure 54

<sup>51</sup> Source: taken from [Lin12] Figure 56





**Figure 88:** Derived graph for the optimization of a two dimensional problem with the *Greedy Ants* approach. The x-axis is the normalized revolution frequency, the y-axis is the normalized spark-timing. The color scale shows the value of the objective function. High z-values symbolizes very good data points<sup>49</sup>

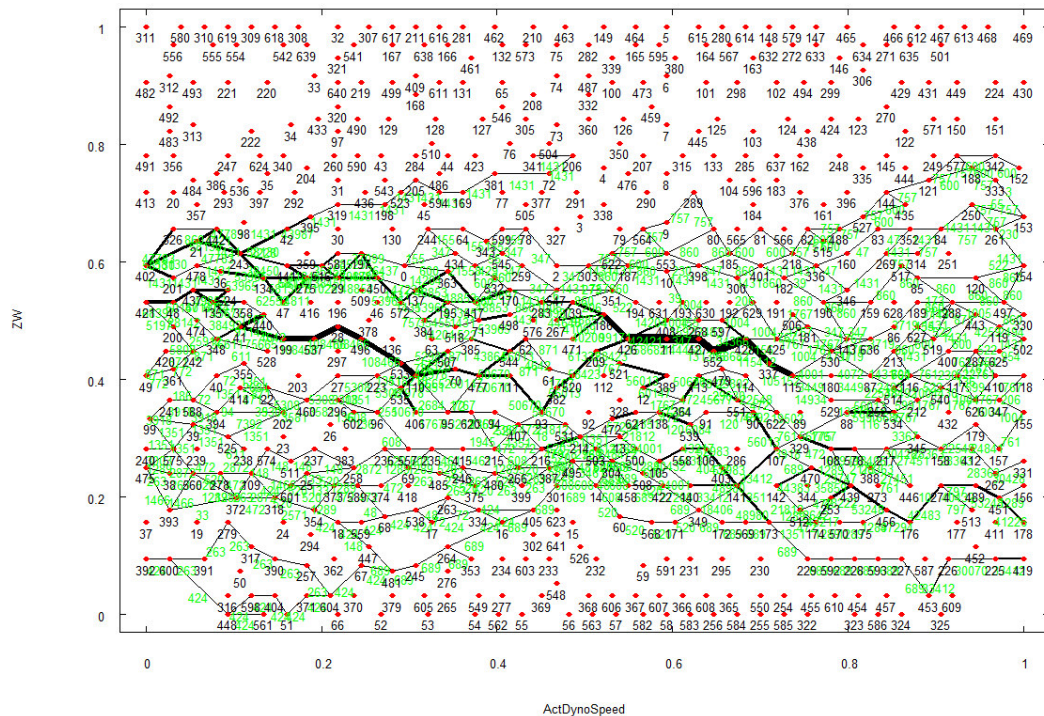


Figure 89: Resulting imprinting of graph with the Greedy Ants approach<sup>50</sup>

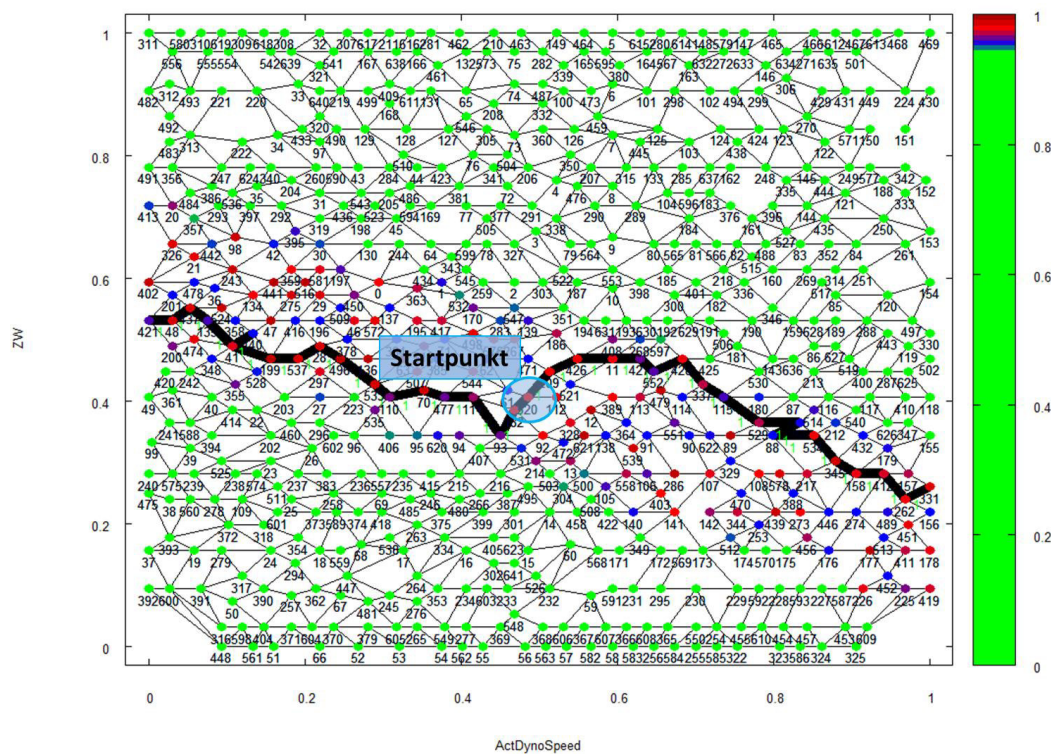
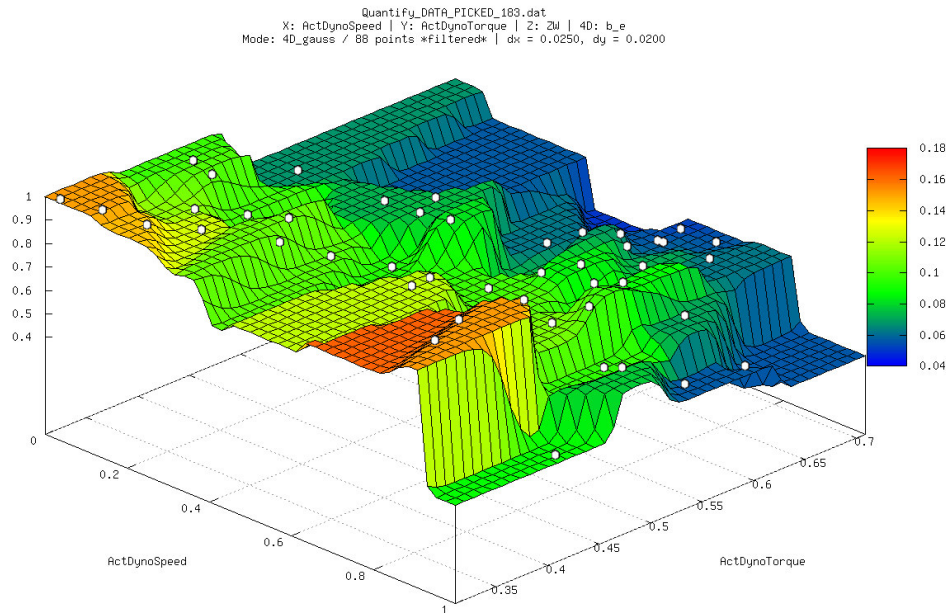


Figure 90: Transient solution of a two dimensional problem with the Greedy Ants approach. *Startpunkt* marks the origin of the dominating *chief* ant<sup>51</sup>



the objective function, by reducing the importance of the specific fuel consumption and improving the impact of the violated constraint. The re-weighting process happens for single operation points in the solution, only, so the algorithm is able to initialize the evolutionary program with the previous global optimum (For further information please see [Lin12]).



**Figure 91:** Transient solution of a three dimensional problem with the *Greedy Ants* approach. This four dimensional plot shows the chosen spark-timing  $ZW$  for different combination of revolution frequency  $ActDynoSpeed$  and engine torque  $ActDynoTorque$ . The color scale represents the normalized specific fuel consumption<sup>52</sup>

### 5.3.3.3 Conclusion of the Greedy Ants run

The previous paragraph summarize the results of P. Lind et al. [Lin12]. Now, we are able to determine the differences and the similarities of the *Greedy Ants* approach in comparison to the approach of this work.

The *Greedy Ants* approach optimizes engine maps with an evolutionary program. That is why this kind of algorithms depend strongly on it's parametrization. One is able to define, the number of scout ants per chief ant, the number of chief ants, local imprinting and global imprinting factors and much more. The creation of the graph contains the tessellation of all data points with respect to its Voronoi cells. Furthermore, it is not possible to proof the correctness of the global optimum. The global optimum is defined by the imprinting on the graph of the predominant clan of ants. Furthermore, the algorithm produces transient solutions, only. This is caused by the optimization directly on the graph, that includes the tolerance constraint, directly. The algorithm is able to optimize on the basis of many criteria, but each criterion increases the necessary run-time strongly.

The *Greedy Ants* approach is the third stage of the *AmmOC* project and the direct predecessor of the algorithm of this work. During the work of P. Lind et al. [Lin12], the project team develops analytic algorithms in order to improve the measurement quality and effi-

<sup>52</sup> Source: taken from [Lin12] Figure 60

---

ciency (see [Lin12] Section 4.3). These algorithms are still applied in the latest optimizer approach. Additionally, this work focuses on the Voronoi tessellation (see Subsection 4.4.1) and the adaptive space compressor (see Subsection 4.3.2). Both methods are essential for the *Integer Linear Program* approach. The *Greedy Ants* optimizer is the first optimizer, which works with the same data basis and complete mathematical description than the *Integer Linear Program* approach. So, this optimizer grants all constraints, which have been defined in this work, i.e. the hard boundary, the integral emission and the tolerance constraints. Moreover, the *Greedy Ants* algorithm constructs complete global optima.

Evolutionary algorithms are a good way to solve complex optimization problems. These example runs are computed on one single computer with 4 cores, but the algorithm is prepared for highly parallel operation. Each *Scout Ant* that serve as local heuristic algorithm is able to run in parallel, without any bottleneck effects. That is why, this algorithm is able to handle a great number of valid data points. The *Integer Linear Program* approach is limited to around 200,000 data points if the computer is equipped with 16 GB RAM. The *Greedy Ants* algorithm is a very early evolutionary stage of a very powerful evolutionary method. So, one is able to improve this algorithm dramatically.

---

## 5.4 Optimization Results

---

This subsection presents the optimization results of both reference optimization problems, i.e. *VTB* (see Subsection 7.2.1) and *AVL\_MM* (see Subsection 7.2.2). In order to study the impact of several actuators, we perform the optimization calculations with different subsets of dynamic actuators.

Each optimization process is divided into four procedures. The first stage is the creation of the stationary solution map. In this case, the optimization problem consists of the objective function (see Equation (119)) and the critical result constraint (see Equation (121)), only. This is applicable for generators, which does not have to satisfy exhaust emission regulations. The second stage includes emission regulations, consequently Equation (120) has to be fulfilled for carbon monoxide, nitrogen oxides, hydrocarbons and soot. The third stage is given by the optimization of the transient engine map without emission regulations. So Equation (119), Equation (121) and (122) have to be maintained. The final stage adds the emission constraints to the transient engine maps. The creation of transient engine maps is the result of the emission basis calibration process.

In the following subsections, we study the *VTB* with focus on the optimization of the *Air path*, which is given by the optimization of cam phase shift, air valve and waste gate actuator. These actuators have to satisfy low gradients (see Equation (122)) on the solution map. Since the *VTB* does not compute exhaust emission values, we have to skip stage 2 and 4. The aspect of raw emission application is studied in the optimization of the *AVL\_MM*.

The linear integer solver *CPLEX* is configured with default settings, except for the maximal gap between *Best Bound* and *Best Integer* (*mip tolerances mipgap*) set to 1 %, emphasis set to *Integer Feasibility* (*emphasis mip*) and *Memory safe* (*emphasis memory*).

---

### 5.4.1 Transient Optimization of AVL Engine Model

---

#### 5.4.1.1 Stage 1: Stationary Solution Map

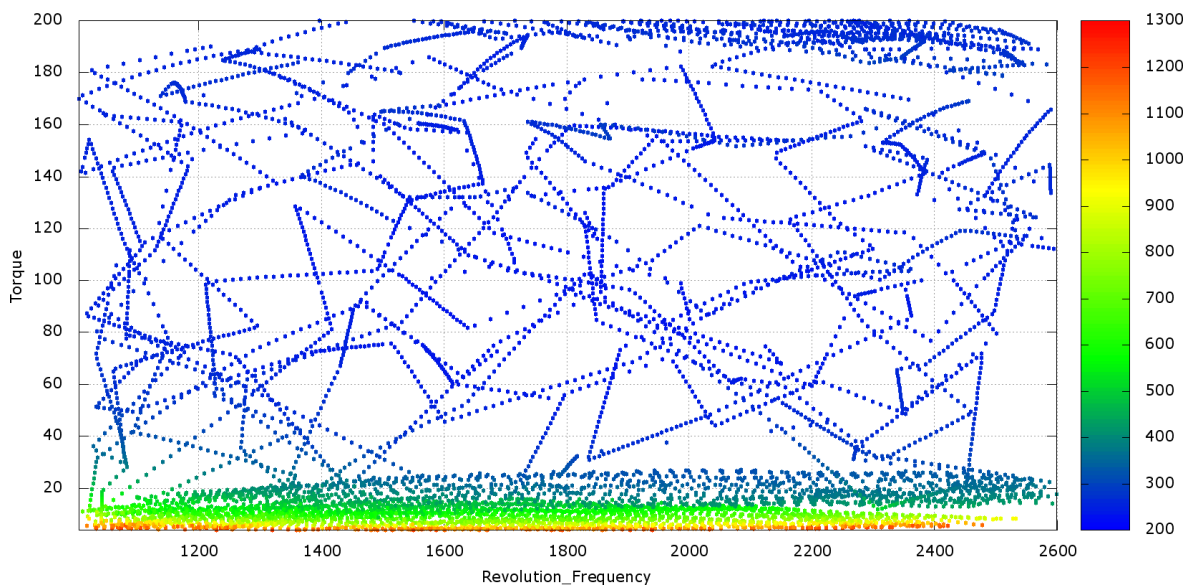
**Stage 1: Setup:** A good foundation for an optimization is a well defined low and high torque border region, which means, that we have to limit certain actuators, in order to focus on these two specific regions. A compression ignition engine has a strong linearity of it's generated *Torque* to the actuator *Injected Fuel Quantity*. That is why we limit the *Injected Fuel Quantity* from up to  $60 \frac{g}{h}$  to  $10 \frac{g}{h}$  in order to measure the very important low torque region. This region is measured for 6 hours (all time specifications are given in real-world measurement time, not calculation time). During this run, *Revolution frequency* and *Injected Fuel Quantity* is varied. The static actuators are set to values that support the creation of low torque operation points, i.e. *Main\_Timing* = 0.0 °CA, *Rail\_Pressure* = 295677 hPa, *VTG\_Pos* = 30.0 °, *Pilot\_Mass* = 0.4 g/cycle and *Pilot\_delta\_t* = 1540.0 °CA.

This region is measured for a second time span of 6 hours with slightly modified settings, i.e. *Main\_Timing* = 10.0 and *Rail\_Pressure* = 405677 hPa. Afterwards, the *Injected Fuel Quantity* gets a lower bound to  $50 \frac{g}{h}$ , so that the high torque regions can be measured. The static actuators are set to values, that support high torque operation points, i.e. *Main\_Timing* = 10.0 °CA, *Rail\_Pressure* = 1126537 hPa, *VTG\_Pos* = 85.0 °, *Pilot\_Mass* = 0.4 g/cycle and *Pilot\_delta\_t* = 1540.0 °CA. Measurement time is again 6 hours.

Finally, the lower and upper bound of *Injected Fuel Quantity* are removed, in order to get a picture of the remaining region of the solution map. Therefore, the static actuators are set to middle values. This region is measured for 6 hours, too. After 18 hours, we get a coarse picture of the engine behavior, which is the foundation of the following optimization steps.

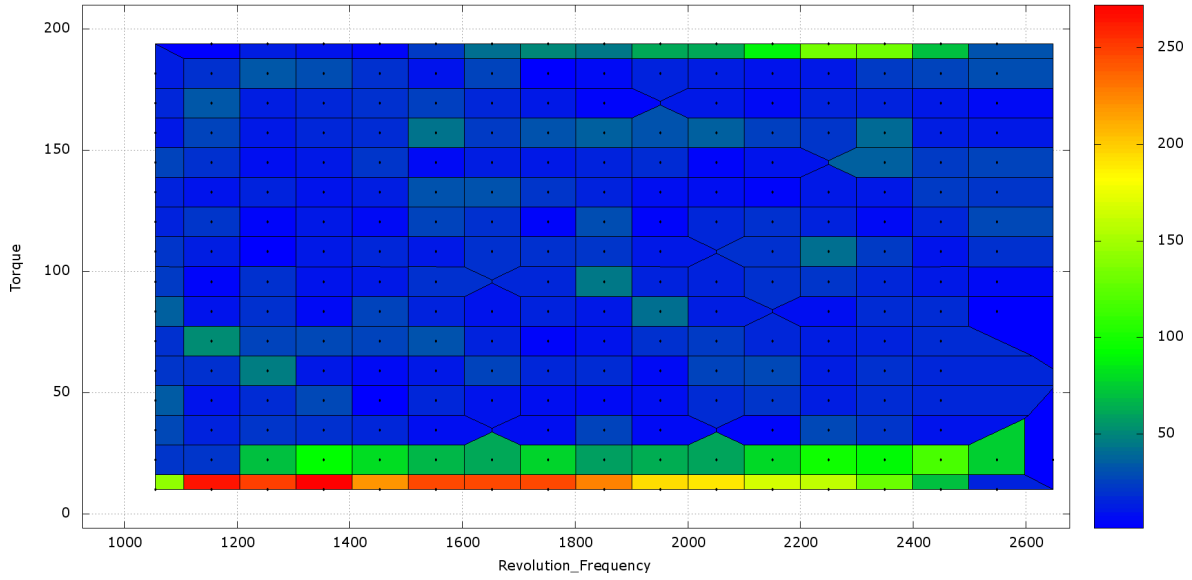
In order to archive an optimization of the specific fuel consumption in each operation point, we run *AmmOC* in *Hunt* mode for *Specific Fuel Consumption*. So, *AmmOC* varies a user defined subset of actuators in order to find better operation point configurations. In this early stage, we add *Air\_Filling*, *Rail\_Pressure* and *VTG\_Pos* to the set of dynamic actuators. During this run, the optimization of *Specific Fuel Consumption* takes 30 hours. Furthermore, the algorithm hunts for holes in the solution. This *Hunt* mode detects holes in the solution and interpolates the actuator settings with respect to the surrounding operation points in the solution. Afterwards, the *Hunt* module derives a small measurement plan and submits the new measurement plan to the test bench. The resulting map is called stationary solution map without emission regulation.

**Stage 1: Results:** Figure 92 illustrate the valid data points during the first stage of the optimization. The axis of this plot are *Revolution Frequency* and *Torque*, so the valid data points are shown with respect to the solution field, which has to be derived. Figure 93 shows the same information, but the valid data points are summed up into the solution field histogram *Sol\_Tree* with 16 times 16 bins for *Revolution Frequency* and *Torque*. As explained in the previous paragraph, we focus on the low and the high torque region. The middle region is resolved with a low, but constant probability. Only a handful of bins are empty, which means, that no data point has been valid or measured in this small region. These *holes* are treated in the following working steps.



**Figure 92:** Valid data points in the end of stage 1. Color scale shows specific fuel consumption in  $g/kWh$ .

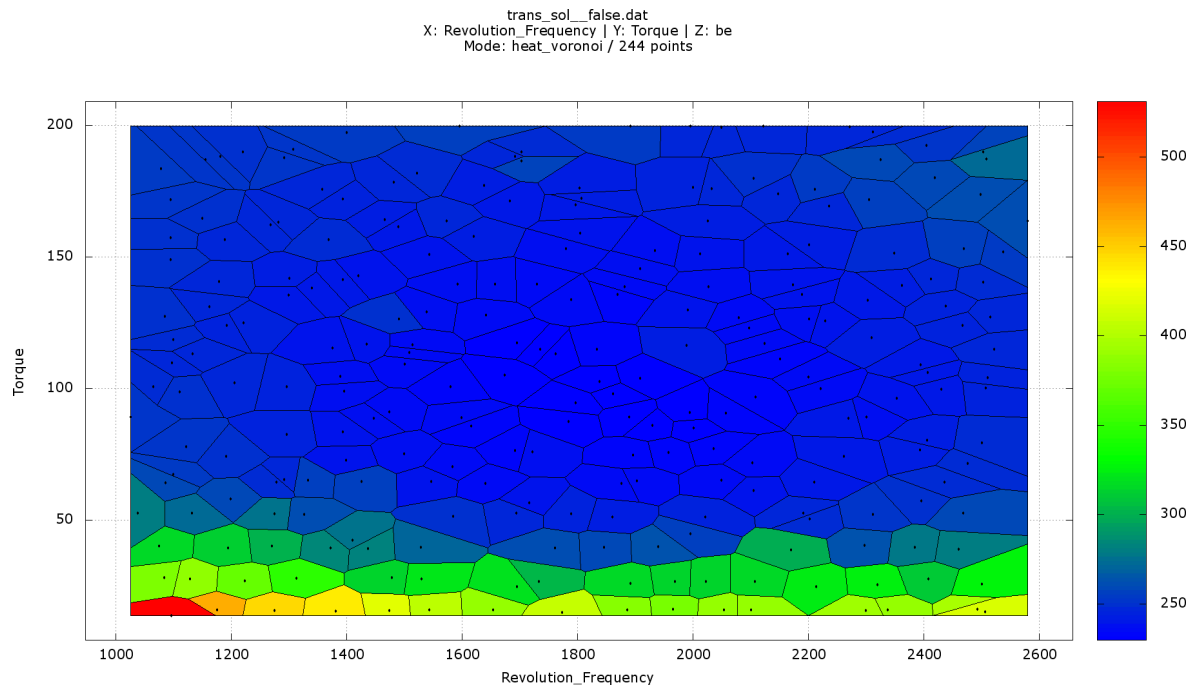
This first stage delivers the first iteration of the solution map for each actuator and each observable. Figure 94 shows the resulting specific fuel consumption for each operation point. This figure is illustrated with colored *Voronoi* sites. The sites are given by the chosen data points for the solution. The advantage of this kind of illustration is the avoidance of any data interpretation. The whole engine map is defined by discrete data points and its *Voronoi* sites. The specific fuel consumption behaves as expected, high values for the low torque values and low values for the high torque region. Figure 95 and 96 are exemplary for two dynamic actuators of this run. The crucial difference between these two figures are the systematic dependency of the chosen values. Whereas the *Injected\_Fuel\_Quantity* shows a strictly pro-



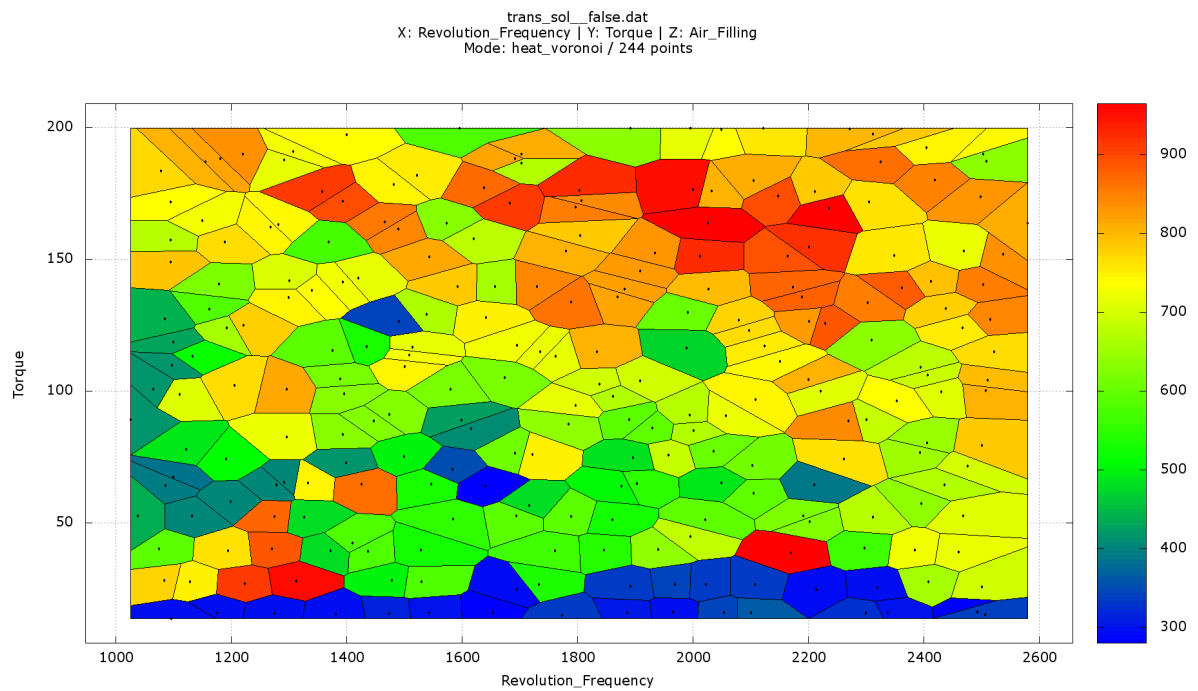
**Figure 93:** Resulting density of valid data points in the end of stage 1.

portional correlation between torque and fuel quantity, Figure 95 seems to be without a trivial relation between the amount of air and torque. However, a dynamic actuator has to maintain its maximal solution gradient in the transient solution as defined in Subsection 4.4.3. If a dynamic actuator does not maintain its maximal gradient constraint, the internal combustion engine will not work probably in the transient mode.

Finally, Illustration 97 shows the selection of data points in the end of stage 1 with respect to their *prey* value (see Equation (9)). In the first stage of the optimization we do not apply any restrictions to the solution, so, the integer solver is able to choose the best specific fuel consumption value in each operation point.

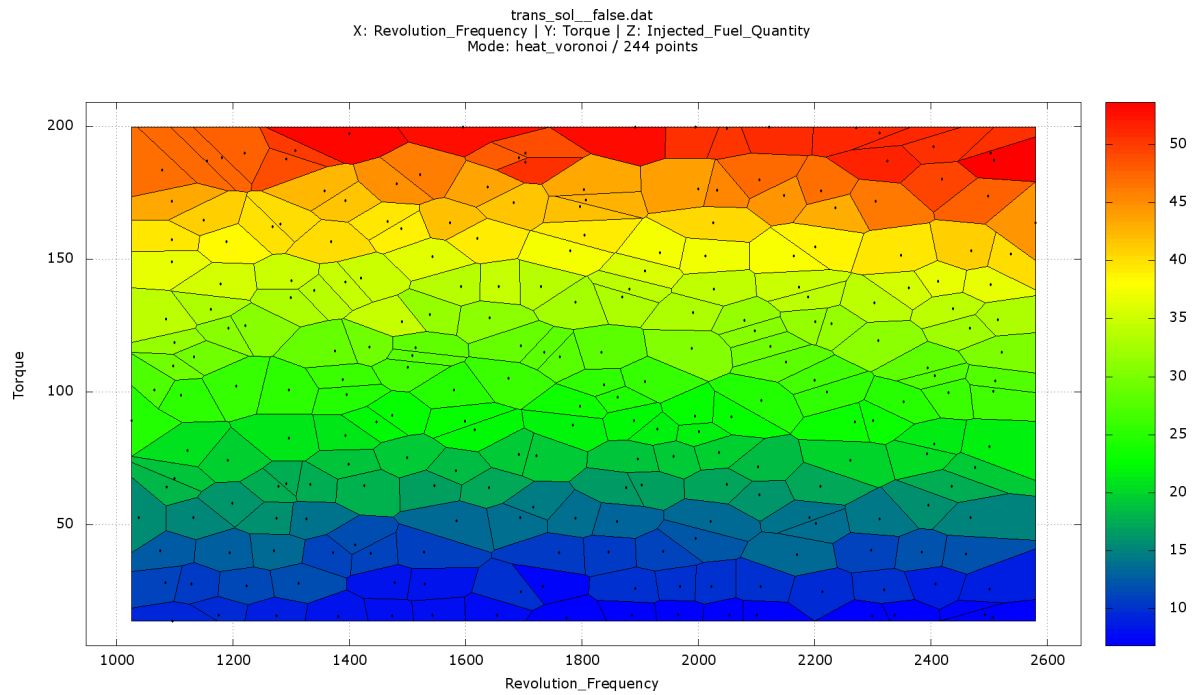


**Figure 94:** Resulting specific fuel consumption ( $g/kWh$ ) in the end of stage 1.

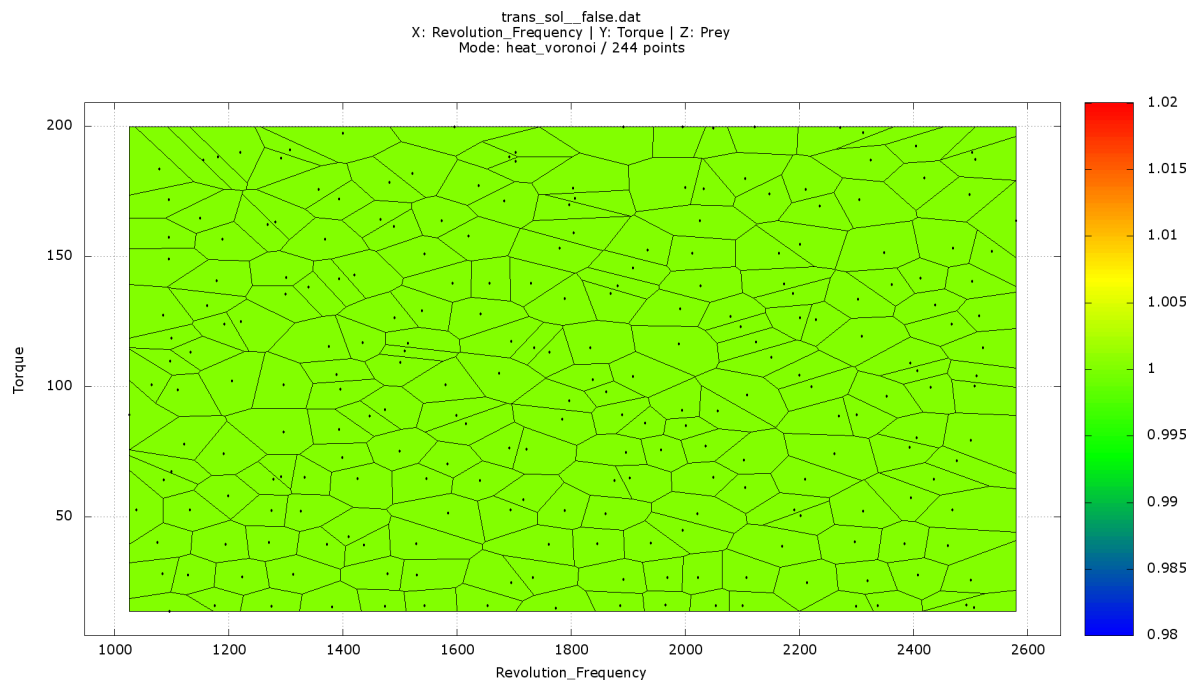


**Figure 95:** Chosen setting for *Air\_Filling* ( $mg/stroke$ ) in the end of stage 1.





**Figure 96:** Chosen setting for *Injected\_Fuel\_Quantity* ( $\text{mm}^3/\text{cycle}$ ) in the end of stage 1.



**Figure 97:** Chosen *Prey* values in the end of stage 1.

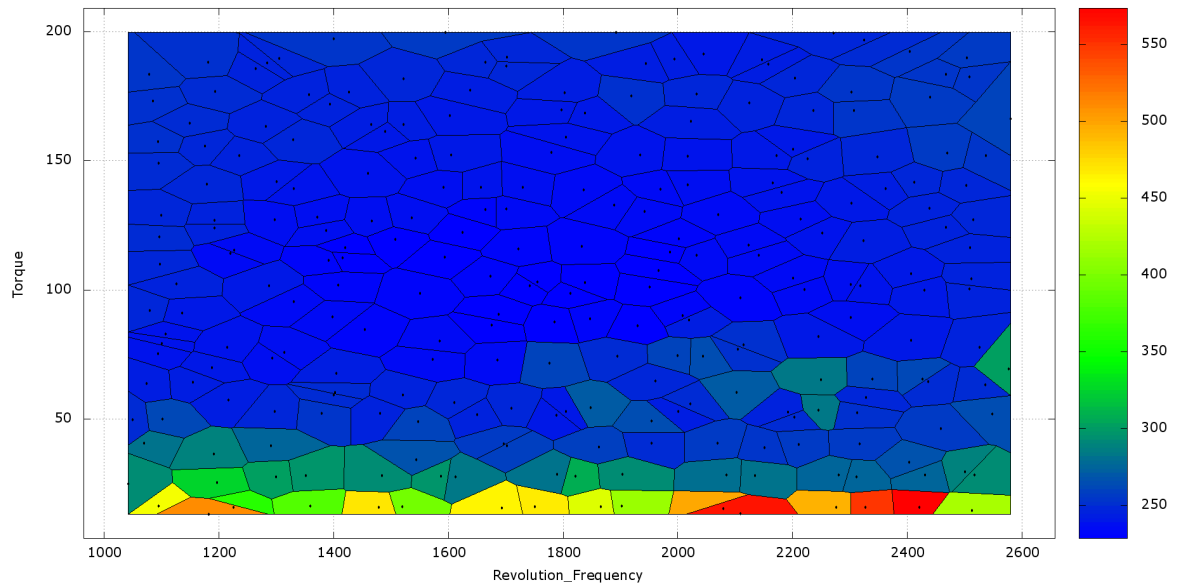
#### 5.4.1.2 Stage 2: Stationary Solution Map with Emission Regulation

The second stage of the optimization is based on the previously derived solution map and its valid data points. This stage aims for emission limits, therefore the algorithm analyzes the solution and marks the parts of the solution that contribute most to the emission integrals. Since, nitrogen oxides are the most problematic exhaust emission species, we focus on these criterion first.

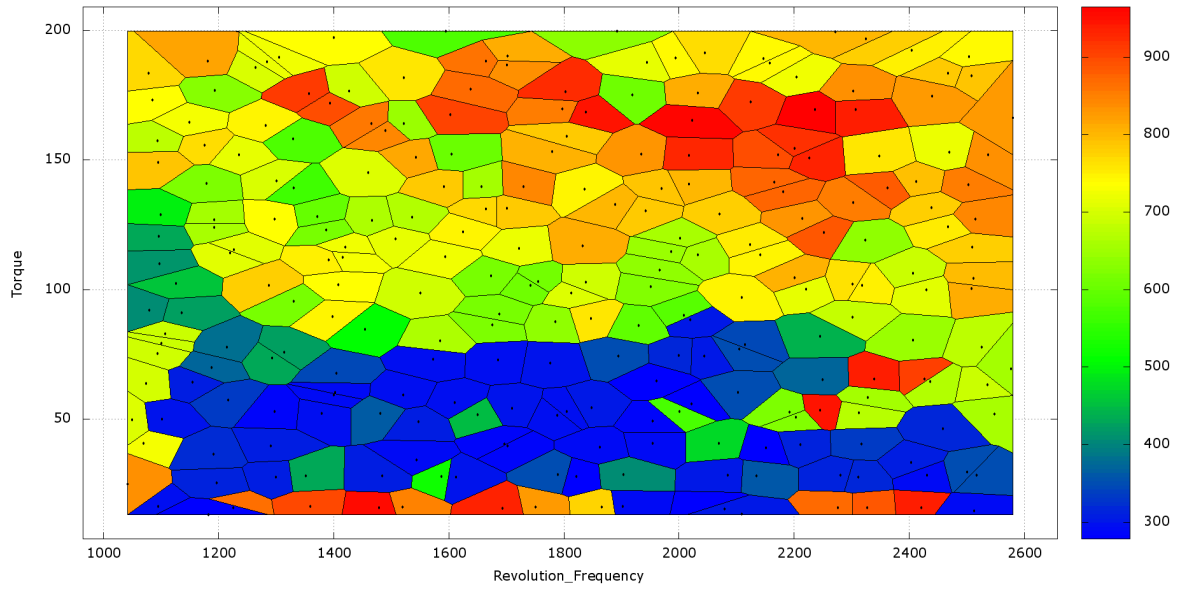
First reference emission test procedure is the new European driving cycle *NEDC*. After the calibration for the *NEDC* has been successful, we continue with the calibration of the more challenging real world driving cycle *RANDOM*.

The second stage starts with an integral value for nitrogen oxides of  $5.635 \frac{g}{NEDC}$ , an integral value for hydro carbons of  $1.885 \frac{g}{NEDC}$  and an integral value for carbon monoxide of  $6.039 \frac{g}{NEDC}$ . The limits for EURO 5 are given in Table 3. The number of filtered data points is fixed to 25000. The adaptive space compressor ASC selection mode (see Subsection 4.3.2) is set to select the best values for nitrogen oxides. After 68.59 hours, we obtain a stationary solution map with EURO 4 norm and 26424 valid data points have been measured.

The EURO 5 norm is accomplished after 88.41 measurement hours. The system measures 34590 valid data points, which are compressed to 25437 data points by the ASC. The average fuel consumption for 100 km is determined to 4.51 liter. The specific fuel consumption for each operation point evolves from stage 1 to stage 2. The specific fuel consumption increases significantly for operation points with very low torque values (see Figure 98). This is clearly visible for operation points below  $90 Nm$ . Due to the additional measurement time, *AmmOC* has been able to find additional data points that improve the specific fuel consumption for some operation points. The modified observables are caused by new actuator settings. In this stage six of eight actuators are varied, but the pilot injection is still ignored. Figure 99 shows the new settings of *Air\_Filling* in the end of stage 2. We recognize the formation of clusters with similar values. The major variation happens in the operation points below 90 Nm of torque.



**Figure 98:** Resulting specific fuel consumption ( $g/kWh$ ) in the end of stage 2.

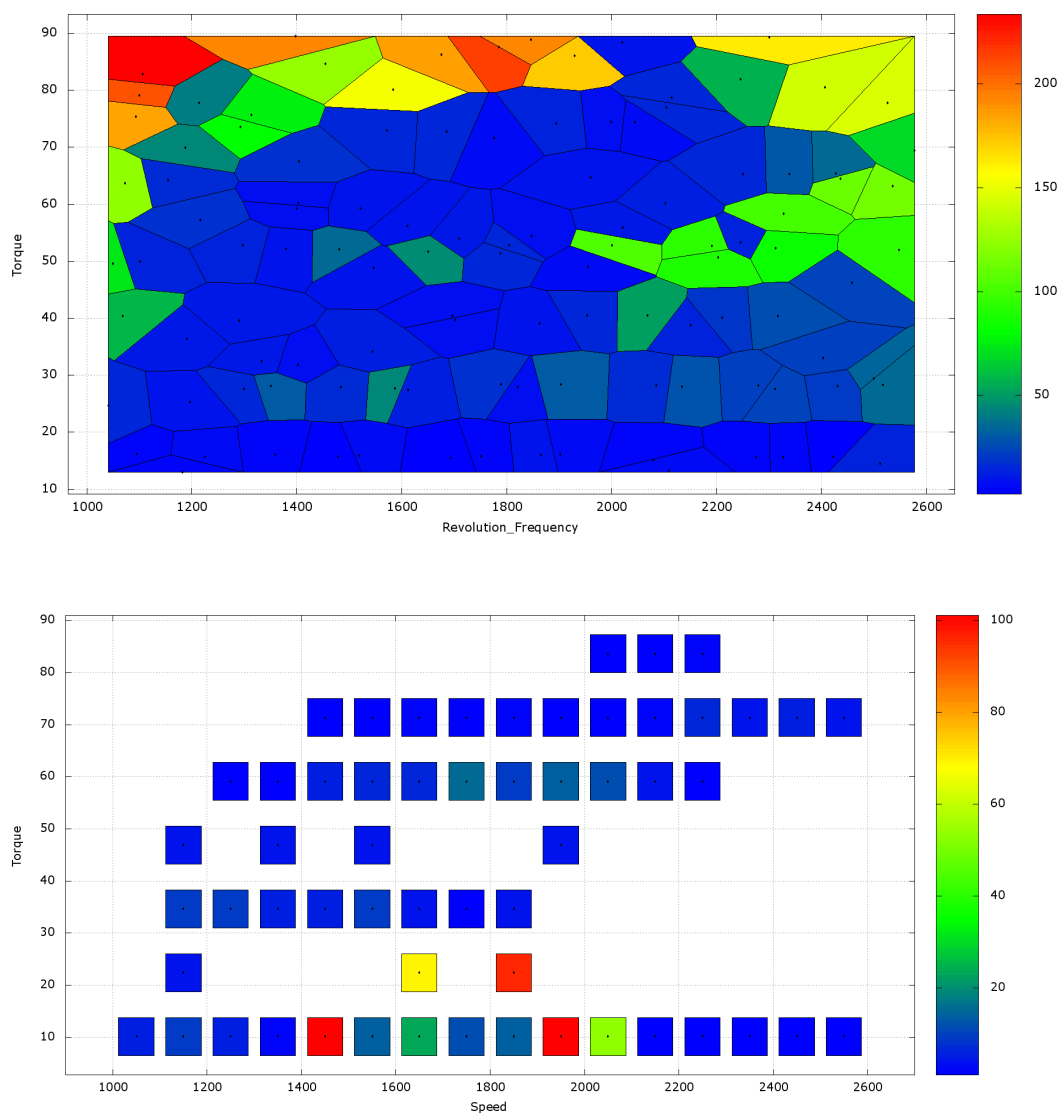


**Figure 99:** Chosen setting for *Air\_Filling* (*mg/stroke*) in the end of stage 2.

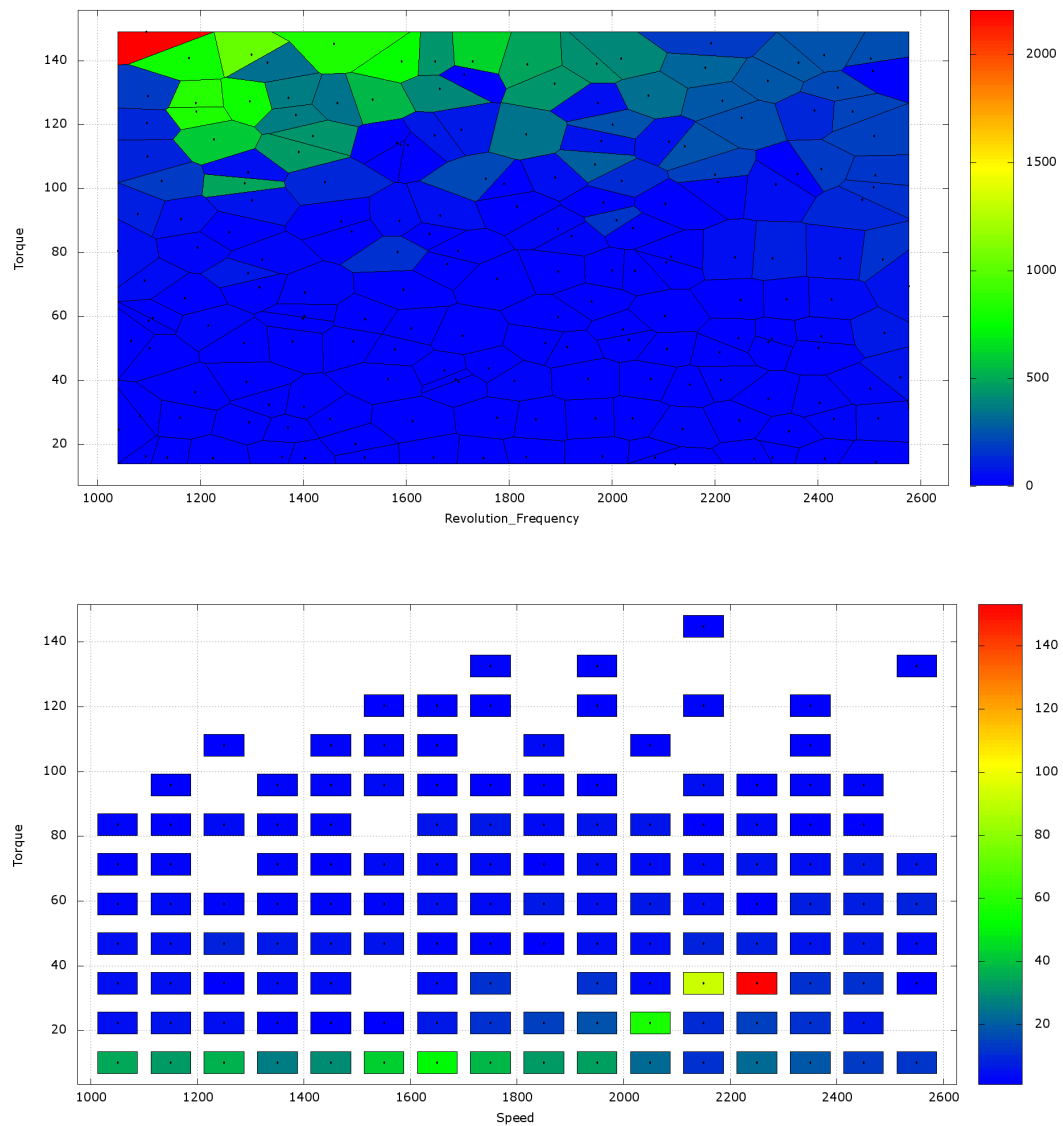
The reason for the modification of the low torque, i.e. below 90 Nm, region is given in Figure 100. The figure in the button shows the weighting table of the *NEDC* (Equations given in Subsection 2.2.3.1). The color of each square displays the duration time in this operation point during the test procedure. Blank regions do not take part in the test cycle at all. The top figure shows a zoomed view on the exhaust emissions of nitrogen oxides in the end of stage 2. We notice clearly, that the solver reduces the exhaust emissions only for the operation points, which take part in the test procedure. This behavior is in the sense of the test procedure, but makes the calibration strongly dependent on the test cycle.

Now, we focus on the new *RANDOM* cycle. It is possible to derive a EURO 4 solution with the new test procedure with the existing data set. A solution map, which maintains a EURO 5 solution has been derived after 107.55 hours of measurement.

The configuration of the actuator maps is quite different. The reason for this becomes obvious in Figure 101. This figure shows the weighting table of the *RANDOM* cycle (button) and the resulting nitrogen oxides emissions after the calibration for the *RANDOM* cycle (top). The weighting table for the *RANDOM* cycle differs from the weighting table to the *NEDC*. The *RANDOM* cycle contains much more operation points that take part in the test procedure, moreover the *RANDOM* cycle includes operation points with torque values up to 145 Nm. We notice, that the integer solver chooses small exhaust emissions values for the operation points that take part in the cycle. The top left corner does not contribute to the cycle, that is why the exhaust emissions are 20 – 100 times higher than in the center of the map.



**Figure 100:** Top: Resulting NOx emissions ( $g/h$ ) in the end of stage 2 (zoom); Bottom: Weighting table of the NEDC.



**Figure 101:** Top: Resulting NOx emissions (g/h) in the end of stage 2 (zoom) after calibration for *RANDOM* cycle; Button: Weighting table of the *RANDOM* cycle.

---

#### 5.4.1.3 Stage 3: Transient Solution Map

The transient solution map is defined as map that satisfies the critical constraint (see Equation 121), the tolerance constraint (see Equation 122) and the stacks constraint (see Equation 118). In principle it is possible to derive the transient solution on the basis of the stationary solution. Yet, the optimization time would be longer, because the transient solution does not have to be similar to the stationary one.

That is why we start with the intention to create a transient solution ab initio. This process is independent of the test procedure. In the first 24 hours of the measurement, we do the basis calibration, in order to define as many operation points as possible in the stationary mode. This means, that the tolerance constraint is ignored in the first hours. Typically, one varies two or three actuators, e.g. *Revolution\_Frequency*, *Injected\_Fuel\_Quantity* and *Air\_Filling*. The advantage of this method is, that five of eight actuators are static, thus their tolerance constraints are not violated per definition.

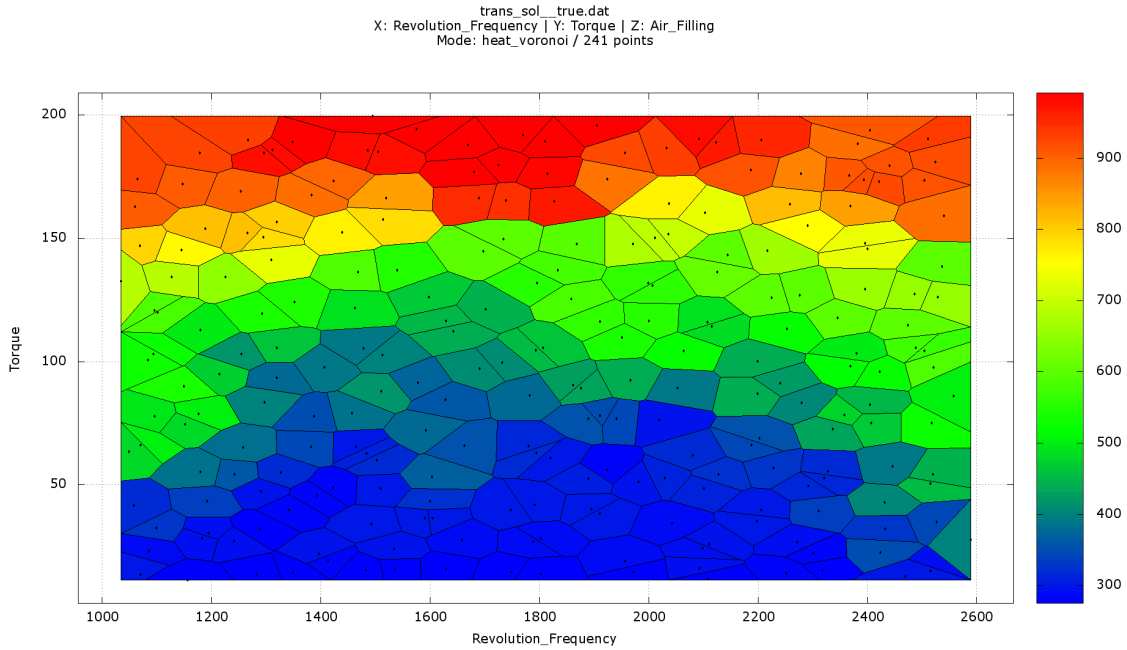
After the basis calibration phase is finished, we switch to the transient calibration mode. Due to the additional tolerance constraints, the algorithm detects new holes in the solution map. These holes are closed by direct measurements in the corresponding region. The expected actuator configuration is derived by a distance weighted average function of actuator configurations of the surrounding data points. In the current example run it takes additional 45.8 hours to close all measurement holes. During this time, the algorithm improved the specific fuel consumption. Therefore, the average fuel consumption for 100 km started with 7.2 liter and improves to 4.65 liter. After all holes are closed, the algorithm focuses on the specific fuel consumption and improved the average fuel consumption for 100 km down to 4.35 liter. The total measurement time is 91.29 hours.

This optimization does not focus on exhaust emissions, that is why the cycle integrals of  $CO$ ,  $HC$  and  $NO_x$  are rather high with 3.68, 1.72 and  $5.26 \frac{g}{NEDC}$  accordingly. This solution map is basis of the following optimization step.

#### 5.4.1.4 Stage 4: Transient Solution Map with Emission Regulation

This optimization step derives the final solution map, which maintains all necessary constraints for a basis raw emission application. Since the tolerance constraint is fulfilled by the previous optimization phase, the algorithm starts to focus on the operation points, that contribute most to the sum of exhaust emissions or  $CO$ ,  $HC$  and  $NO_x$ . Moreover, we discuss in this subsection the question of reproducible solution maps for different emission test procedures. Therefore, we study the predictive power of test procedures on the examples of *NEDC* and *RANDOM*. The validation of a transient solution map is done by test procedures.

Figure 102 shows the setting of the parameter *Air\_Filling*. In direct comparison to the settings of the previous stage (see Figure 99), we notice, that the *Air\_Filling* parameter behaves much more systematically. In particular, the throttle valve stays nearly constant for operation points with equal torque values. If the engine torque is steadily increased the throttle valve opens slowly and conserves the maximal gradient constraint. The previous settings in Figure 99 does not conserve the tolerance constraint. The settings of the actuators after stage 4 are applicable for engine control units *ECUs*.



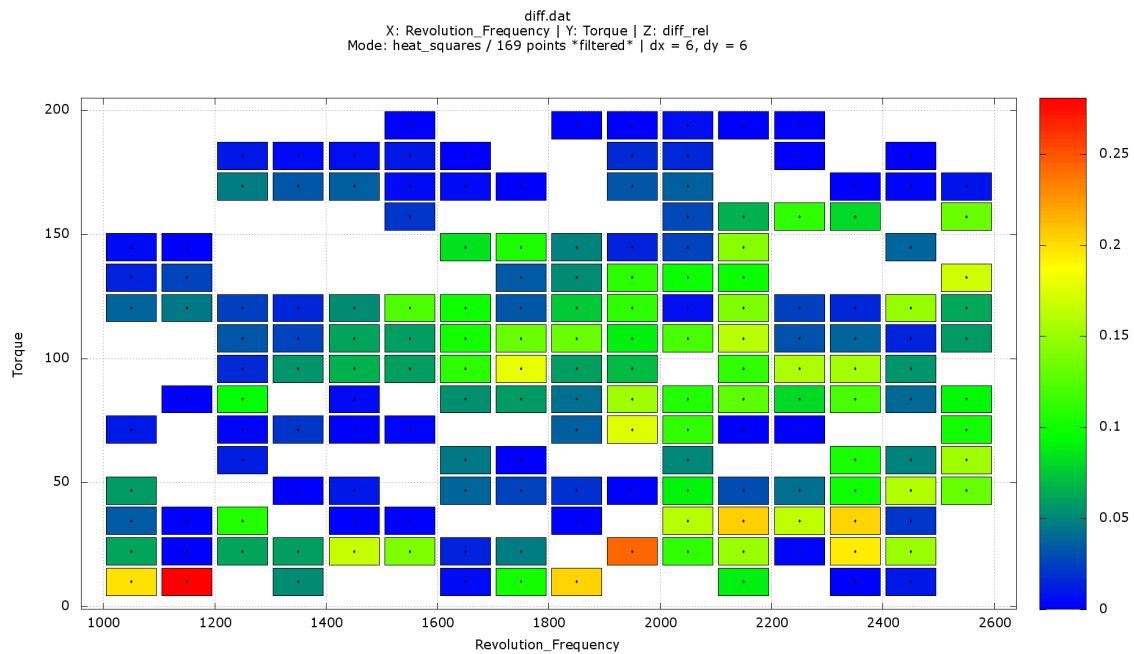
**Figure 102:** Chosen setting for *Air\_Filling* (mg/stroke) in the end of stage 4 in *RANDOM* cycle.

In the next passage, we compare the calibration for the *NEDC* and *RANDOM* cycle. As explained previously, the *NEDC* covers less operation points. Especially, it covers operation points below 90 Nm, the *RANDOM* cycle includes operation points below 160 Nm. Furthermore, the *NEDC* focuses on different operation points than the *RANDOM* cycle. That is why the algorithm chooses different operation point configurations.

Figure 103 shows the difference in the selection of data points in accordance with the specific fuel consumption value for the calibration of both test procedures. The positive values represent higher specific fuel consumption due to the calibration with respect to the *RANDOM* cycle. The figure below (see Figure 104 shows the same information but for the exhaust emissions of nitrogen oxides. The major differences are in the torque region between 60 and 160 Nm. If we consider the weighting tables of *NEDC* (see Figure 100 bottom) and

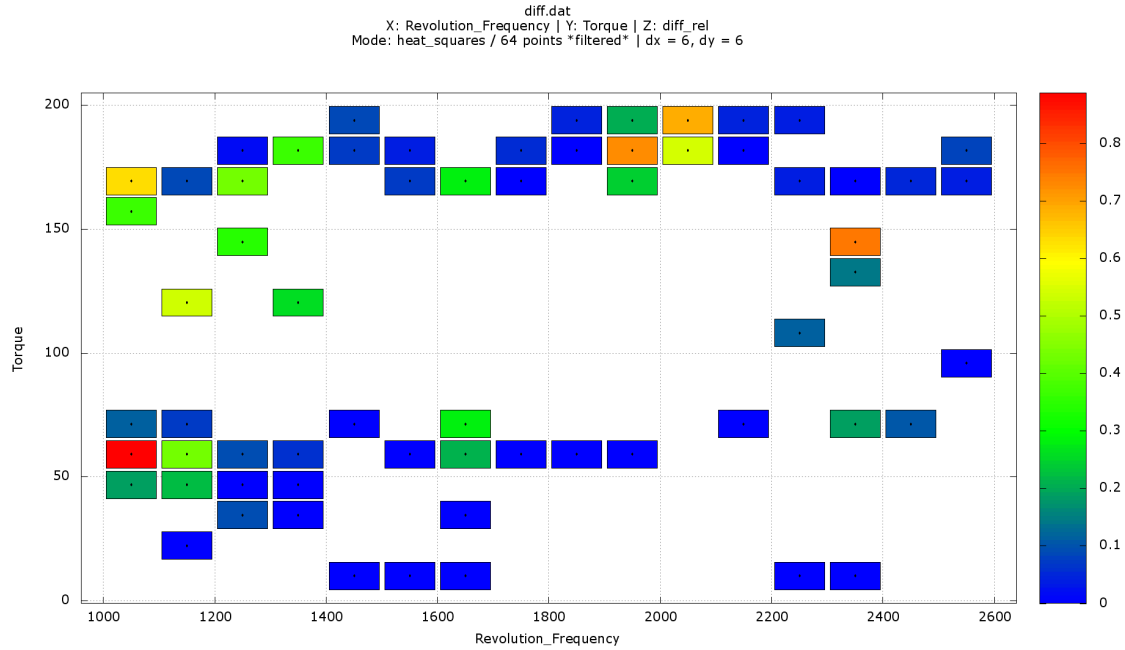
*RANDOM* cycle (see Figure 101 botton), we conclude, that this region is considered in the *RANDOM* cycle, only. Due to the selection of lower nitrogen oxide emission data points the algorithm has to choose worse specific fuel consumption data points. The medial aggravation of the specific fuel consumption is 10 to 15 %. However, the nitrogen oxides emission are reduced significantly in certain operation points.

Due to the number of constraints, the algorithm is not able to take the data point with the best specific fuel consumption of the operation point. This is illustrated in Figure 105. The top picture shows the picked *prey* values of the *NEDC* calibration. The botton picture shows the calibration of the *RANDOM* cycle. The content of these illustrations is linked to Figure 103. The algorithm is able to pick the ideal specific fuel consumption value for the most operation points, which does not take part in the test procedure.



**Figure 103:** Relative difference of the resulting specific fuel consumption for the calibration on *NEDC* and *RANDOM* cycle. Higher values mean higher values in *RANDOM* cycle than in *NEDC*. Operation points with lower values in *RANDOM* cycle than in *NEDC* are blanked out for clarity.



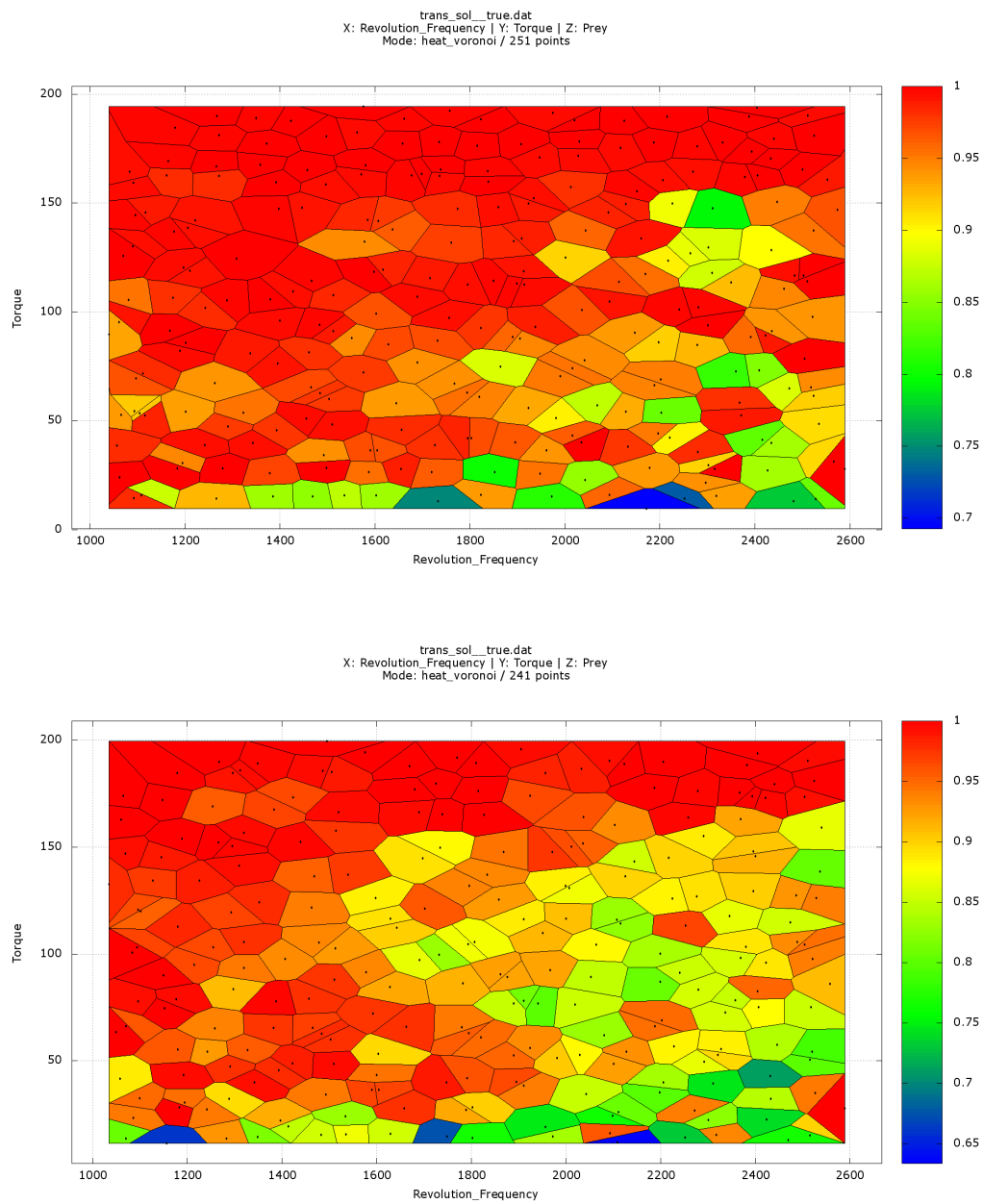


**Figure 104:** Difference of the resulting nitrogen oxides emission for the calibration on *NEDC* and *RANDOM* cycle. Higher values mean higher values in *RANDOM* cycle than in *NEDC*. Operation points with lower values in *RANDOM* cycle than in *NEDC* are blanked out for clarity.

#### 5.4.1.5 Predictive Power of test procedures at the example of *NEDC* and *RANDOM* cycle

The optimization results of stage 4 show, that the final settings of the major actuators depend mainly on the based test procedure, i.e. on the quantification weighting table. The newly announced test procedures, which have to reflect real world driving profiles, differ from the established new European driving cycle. The differences have been discussed in the previous passage, so this passage focuses on the setup of the *RANDOM* cycle. Similar to the *NEDC*, the *RANDOM* cycle consists of operation points that do not contribute as much as some few operation points, e.g. the operation point ( $1950\text{min}^{-1}$ ,  $10\text{Nm}$ ) contribute much stronger than the operation point at ( $2250\text{min}^{-1}$ ,  $10\text{Nm}$ ). The spread among the operation points is very pronounced. This is one major reason why the final calibration solution differs essentially for different test cycles.

In order to get a more reliable solution with a good prediction power, one should reduce the spreading of the contribution times of all operation points. The weighting of the operation points reflect aspects of statistical driving profiles. This generated *RANDOM* cycle include path pieces of city short-trips, rural short-trips and highway short-trips. The spreading can be reduced, if the calibration is not based on one single *RANDOM* cycle, but a statistical evaluation of many *RANDOM* cycles. Alternatively, the calibration is based on a *conservative RANDOM* cycle. A *conservative RANDOM* cycles combines several *RANDOM* cycles and is given by the maximal weighting value of each operation point. In both cases, the spreading is decreased, so the prognosis would be more reliable.



**Figure 105:** Comparison of picked prey values for *NEDC* and *RANDOM* cycle. Top: Picked prey in *NEDC*; Button: Picked prey in *RANDOM* cycle.

---

## 5.4.2 Transient Optimization of VTB

---

In this subsection, we optimize the virtual test bench model *VTB* (see Subsection 4.2.6). The challenge of optimizing this model is quite different from the previous optimization. The previous engine model is based on measured data. These data points have been used to create polynomial fit functions for each observable. Therefore, the systematic behavior is quite smooth. The current engine model is based on numerical functions and numerical integrals (see Subsection 4.2.6), that is why the behavior is much more unsteady, the system behavior is not smooth over the whole range. An additional difference is based in the engine itself. The AVL engine model is at the basis of a diesel engine. The current model is based on a turbocharged spark-ignited engine equipped with camshaft phase actuators. Relating to the linear integer program, we have to fulfill a different and much bigger set of maximal gradient constraints. Despite to the diesel engine, the spark-ignited engine does a lot of modifications to the *air path* of the system. The *air path*, i.e. the supply and removal of air in and out of the combustion chamber, is the slowest physical process inside the internal combustion engine.

The dominating tolerance constraint in the diesel model is the *Air\_Filling* maximal gradient. The major tolerance constraints in the spark-ignited engine are the air valve *Alpha*, exhaust gas regulation valve *AGR* and both phase actuators of the intake and exhaust camshafts.

Because this model does not provide any exhaust emission values, the optimization is focused on the specific fuel consumption values, only. That is why we skip the second and the fourth stage of the optimization. As in the previous optimization, we start with the stationary solution without emission regulation. After this, the optimization continues with the transient solution without exhaust emission regulation. Similar to the previous optimization it makes sense to consider these two optimization steps not as sequence of steps, i.e. if the requested solution is a transient one, one should start with the transient optimization directly. So, the user is able to save a lot of test bench measurement time.

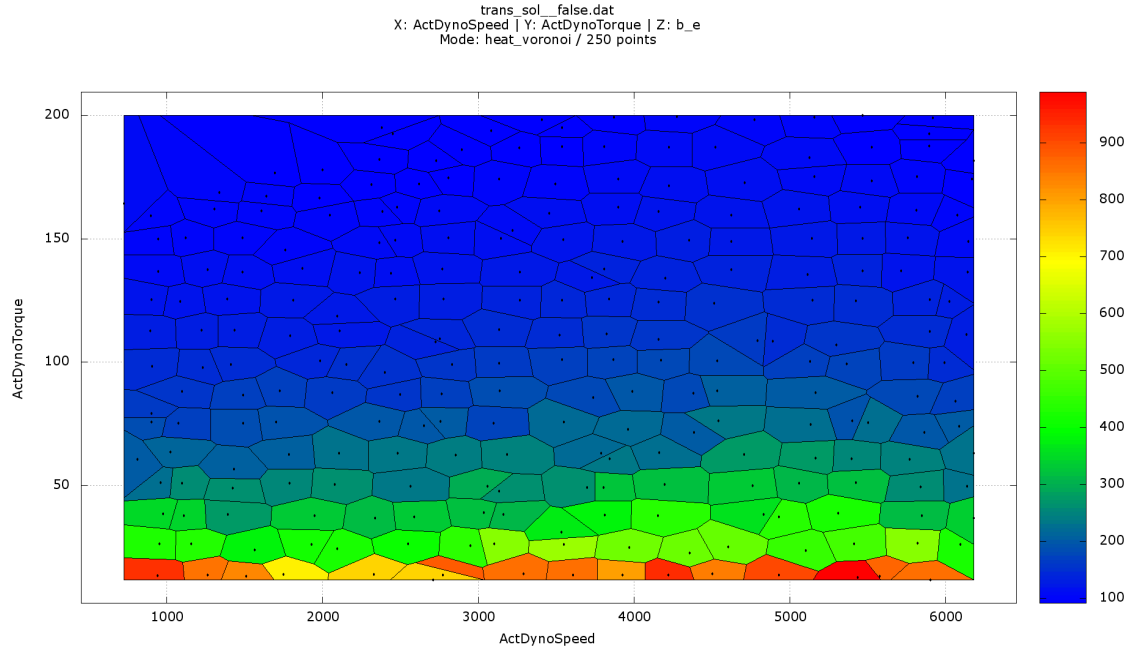
### 5.4.2.1 Stage 1: Stationary Solution Map

The first calibration is based on the *NEDC*. This information affects the calculated mean fuel consumption, only, since exhaust emissions are neglected. After 12.18 hours of measurement we obtain the first stationary solution map without holes. The average fuel consumption per 100 km is 8.45 liter. The mean fuel consumption can be improved after additional 75.39 hours to 8.17 liter per 100 km.

The measurement is continued to a total measurement time of 326.03 hours, however the specific fuel consumption values of the operation points that contribute does not improve any further. But, the algorithm is able to improve other operation points, which do not contribute to the *NEDC*. After 75.39 hours of measurement time the sum of all specific fuel consumption values is 79085 g/kWh. In the end of the optimization run, the sum decreases to 66767 g/kWh. This is an improvement of 15 %. The specific fuel consumption for each operation points is illustrated in Figure 106. The main contribution to the mean fuel consumption comes from the 10 Nm line. Furthermore, the *NEDC* stresses operation points with 10 Nm of torque (see Figure 100 button). Since, the *RANDOM* cycle delivers a much more realistic driver profile, we continue the analysis with the calibration at the basis of the *RANDOM* cycle.

After 67.8 hours we derive a stationary solution without any holes. The mean fuel consumption decreases to 7.2 ltr per 100 km. The optimization process terminates, if the algo-

rithm finds no new local optimum during 6 hours of measurement time. So, the algorithm terminates after 129.14 hours. The results are discussed in the following paragraph.

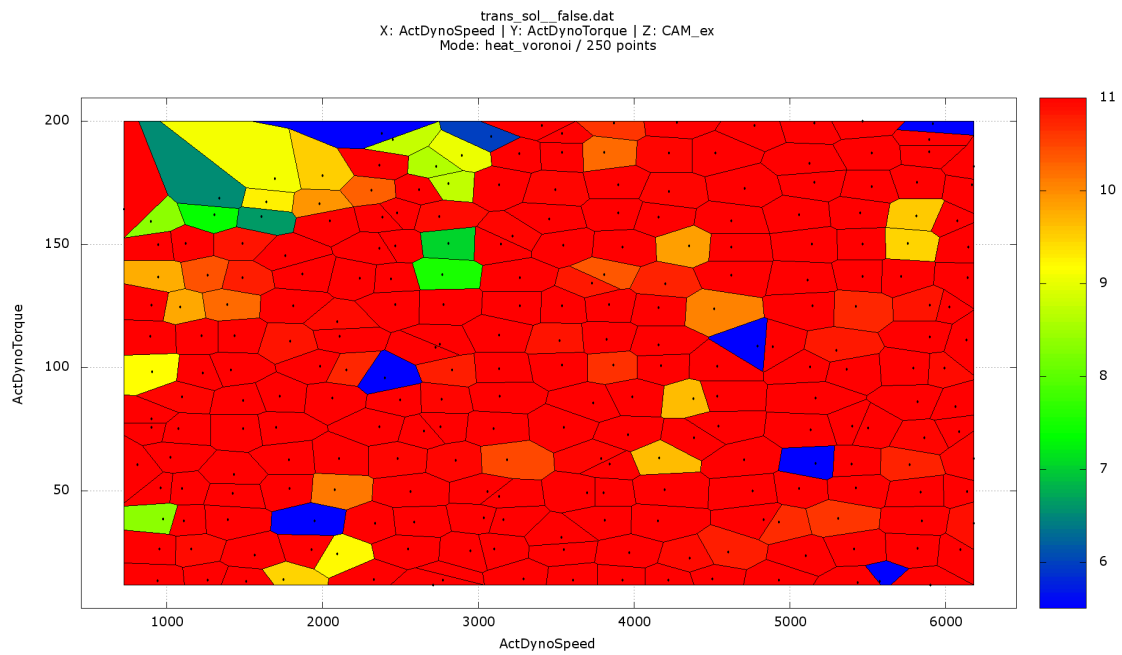


**Figure 106:** Resulting specific fuel consumption ( $g/kWh$ ) of stationary solution (*NEDC*)

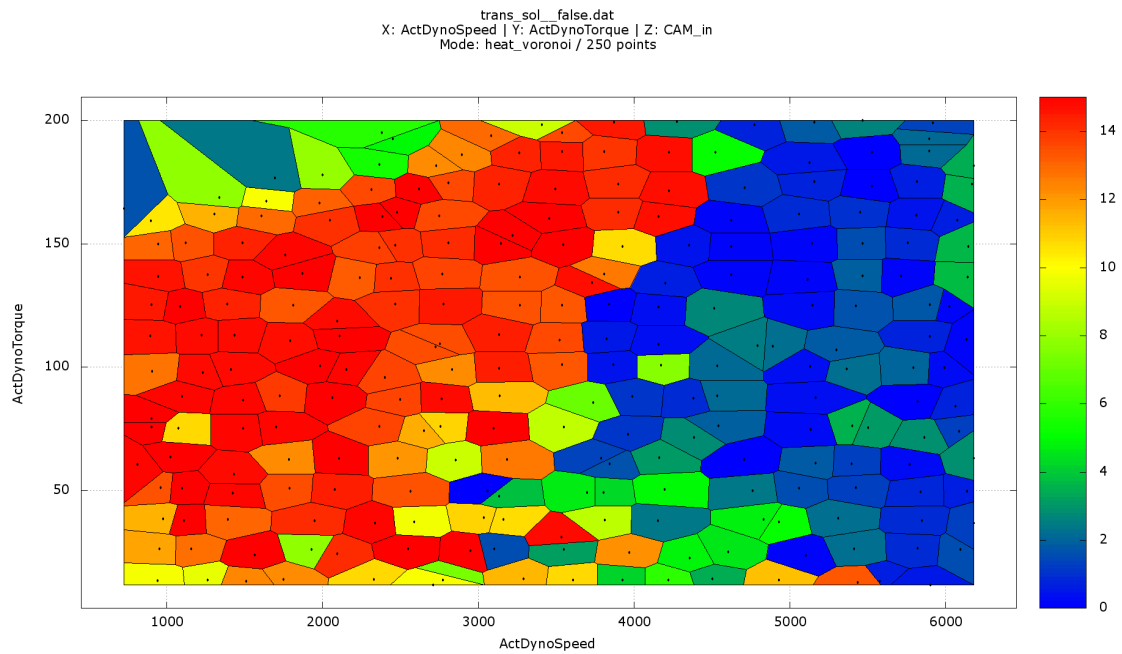
Figure 107 and Figure 108 show the settings of the exhaust and intake camshaft actuators. The phase settings of the intake cam shows a clear behavior. For small revolution frequencies, the intake valve opens as late as possible. For higher revolution frequencies, the valve opens as early as possible. The phase angle of the exhaust valve seems to have a nearly constant behavior at late timing values. These settings are for a stationary solution, only. Some points switch from 0 degree camshaft to 10 degree camshaft in the neighbor operation point, which would violate the tolerance constraint. The mean fuel consumption for 100 km has been derived to 6.64 liter. The settings of the actuators are quite similar. The difference in the mean fuel consumption is caused by the selection of operation points that contribute to the average fuel consumption. Furthermore, this stresses the point, that different test procedures causes different mean fuel consumption values.

Furthermore, we optimize this engine model with the *full factorial* method (see Subsection 3.1.2). So, we are able to compare the optimization quality and speed of this method to our *Dynamic Testdrive* method. The *full factorial* run varies all six actuators, *revolution frequency* and *air valve angle* are resolved with 16 supporting points, the other actuators are resolved with 4 supporting points. The whole measurement time is 4551.11 hours.

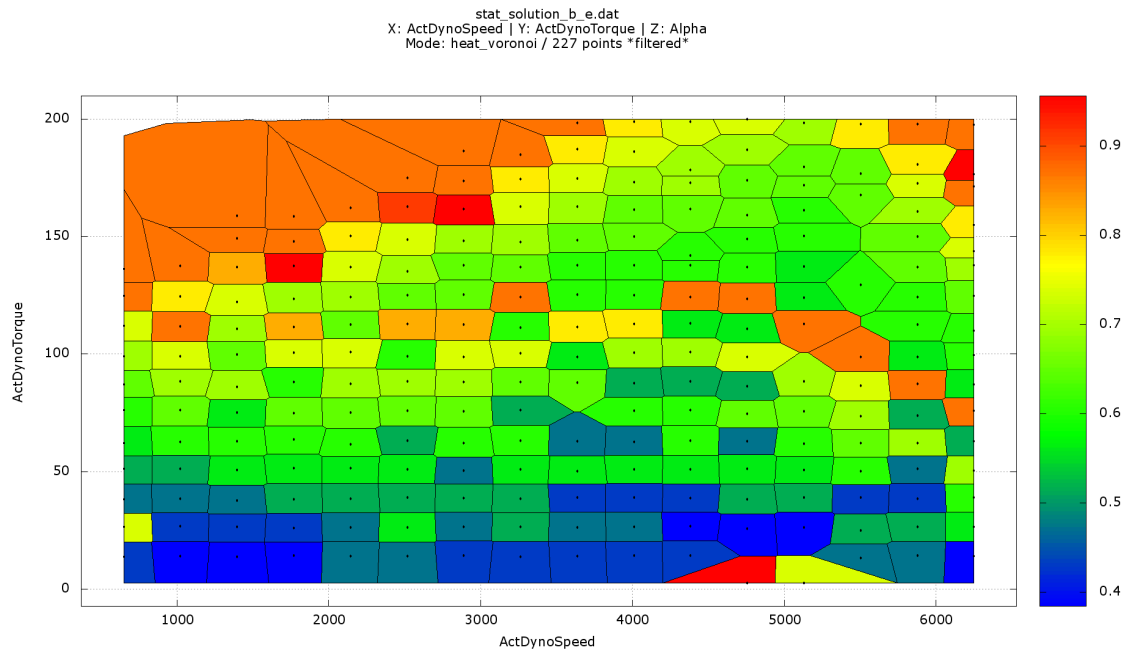
The resolution is four times higher in Figure 109 than in the Figures 111, 112 and 110. The resolution of the actuator settings for camshafts and spark-timing seems to be insufficient. On the other hand, the total measurement time is notably high. This illustrates the biggest disadvantage of the *full factorial* method. Most of the time the resolution is insufficient to resolve the system behavior, but however the total measurement time is much too high. However, the *full factorial* method detects the global systematic behavior of the engine. The intake camshafts have to open lately for low revolution frequencies, and early for higher revolution frequencies. The transition between both regimes is resolved poorly. The settings



**Figure 107:** Settings for exhaust camshafts of stationary solution (*RANDOM*)



**Figure 108:** Settings for intake camshafts of stationary solution (*RANDOM*)

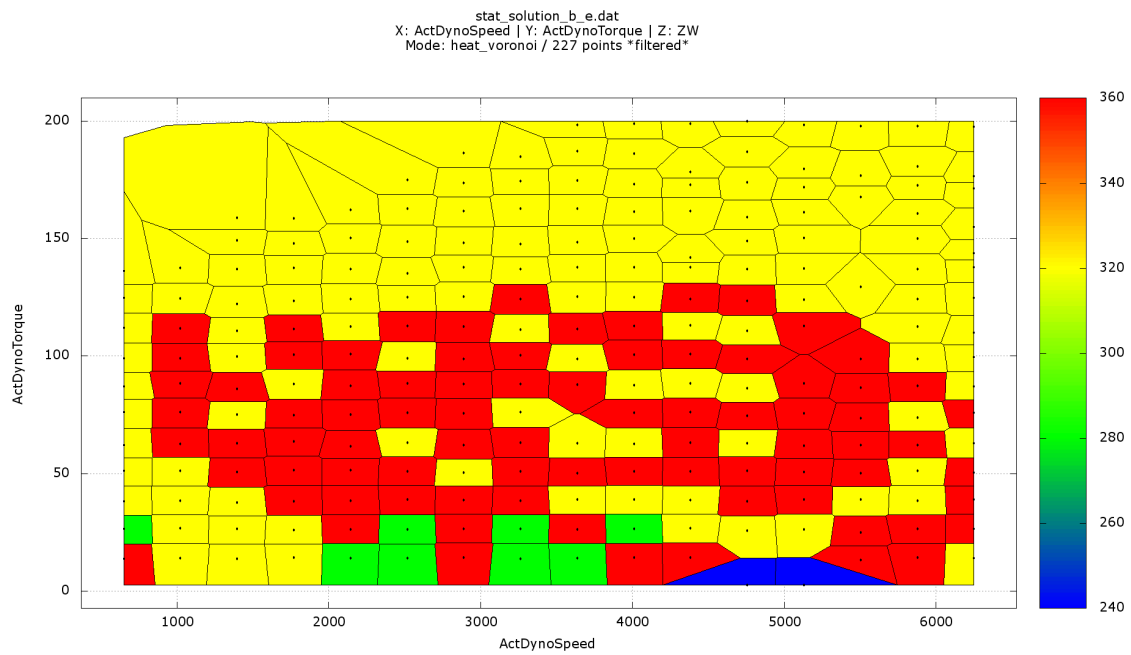


**Figure 109:** Full Factorial run: Settings for air valve angle of stationary solution (RANDOM)

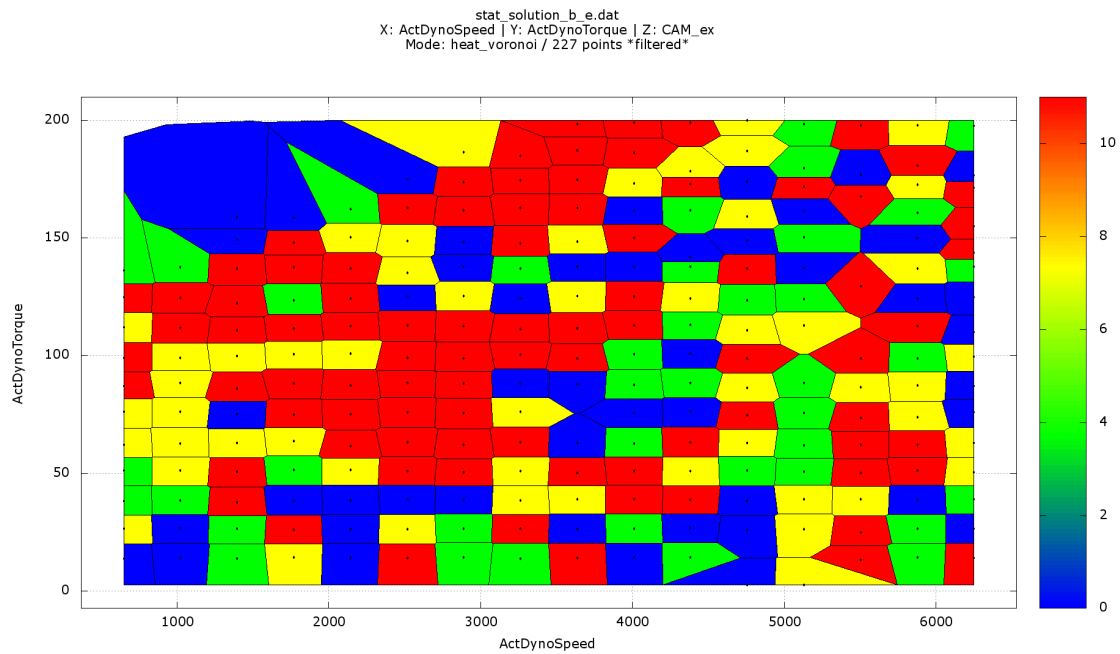
for the spark-timing and the phase of the intake valves seems to be without any systematic behavior.

The *full factorial* method is applied in order to test the found optima of stage 1 of the optimization process. It is hardly possible to proof, that the *Dynamic Testdrive* module finds the global optimum in a given amount of time, that is why we compare the found optimum with a very detailed and unprejudiced method. The mean fuel consumption per 100 km is computed to 6.96 liter by the *full factorial* method. All specific fuel consumption values sum up to 75341g/kWh. This is around 12 % higher than for the *Dynamic Testdrive* run.

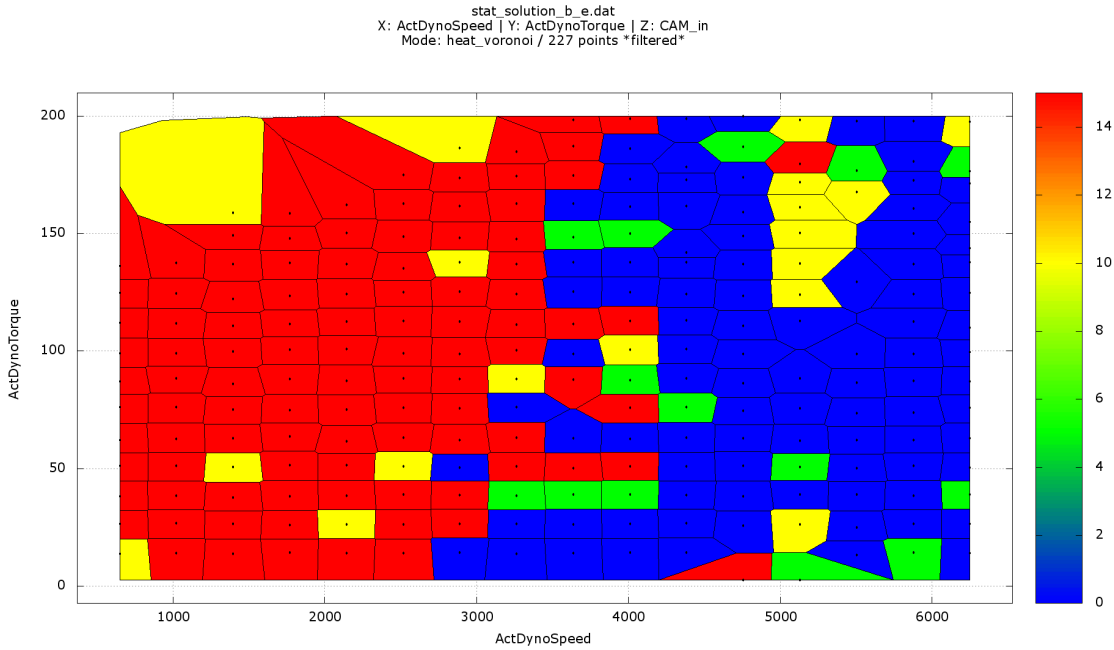
So, we can conclude, that both methods detect a similar global systematic behavior. The derived mean fuel consumption is comparable, but the *Dynamic Testdrive* method is able to find a lower mean fuel consumption value. The run-times are very different. The full factorial method takes longer than 1000 measurement hours, therefore, this method is not reasonable in efficient applications. The measurement time of the *Dynamic Testdrive* module is circa 200 measurement hours.



**Figure 110:** Full Factorial run: Settings for spark-timing of stationary solution (*RANDOM*)



**Figure 111:** Full Factorial run: Settings for phase of exhaust camshaft of stationary solution (*RANDOM*)



**Figure 112:** Full Factorial run: Settings for phase of intake camshaft of stationary solution (*RANDOM*)

#### 5.4.2.2 Stage 3: Transient Solution Map

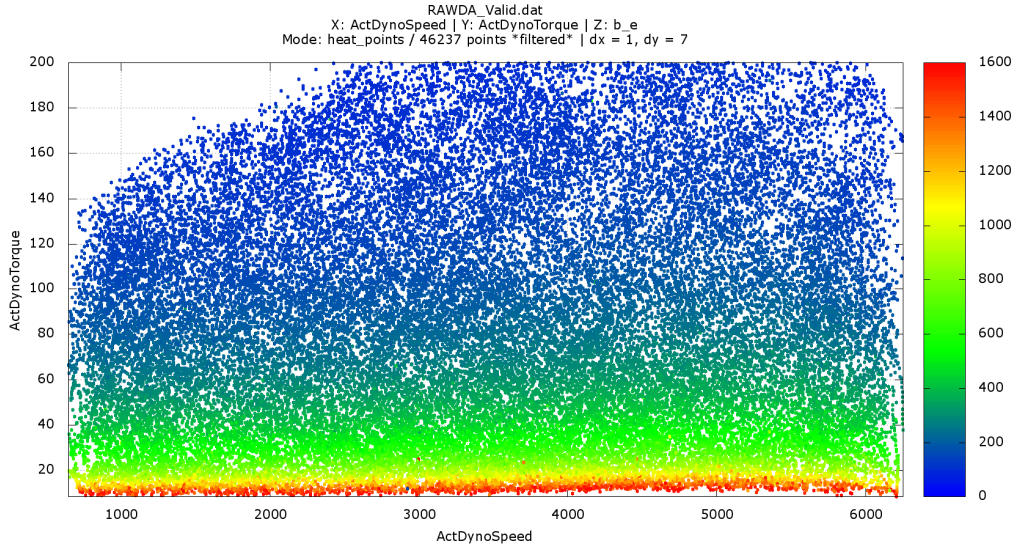
The optimization process for the transient solution map takes 253.05 hours on the test bench. The optimization starts directly on the basis of the *RANDOM* cycle. As in the previous run, we start to vary *revolution frequency*, *alpha* and the *spark-timing*. So, we get a complete map of all operation points (see Equation (1)). This basis calibration takes 21.11 hours. The mean fuel consumption is 7.59 ltr per 100 km. This solution does not conserve the tolerance constraints.

In order to derive the first transient solution map, it is necessary to vary the phase of the intake camshaft, too. The phase of the *exhaust camshaft* and the *exhaust gas regulation valve* stays fixed at  $0^\circ\text{CA}$ . So the exhaust cam opens as early as possible, consequently no burnt gas is mixed into the fresh air of the combustion. After 2.5 hours, we derive the first transient solution map. The corresponding fuel consumption is 7.73 liter per 100 km. Afterwards, all actuators are able to vary, so after 253.05 total measurement time, it is possible to reduce the fuel consumption to 7.05 liter per 100 km.

Figure 113 shows the distribution of valid data points with respect to revolution frequency (*ActDynoSpeed*) and engine torque (*ActDynoTorque*). The color scale shows the specific fuel consumption. we notice, that the low torque region, i.e. operation points with smaller torque than 40 Nm is resolved with a higher point density. Furthermore, the engine torque seems to be bounded for low revolution frequencies, i.e. below 2500 round per minute. In general, all operation points are covered with a similar point density. While this optimization run, we reduce the measurement frequency to 1 data point per second. That is why, the total number of valid measurement points is 46237, only.

Figures 115 and Figure 114 show the critical engine pressure and the critical temperature. The critical temperature is defined in Subsection 2.2.2.1. The upper limit of this sensor is 850K. We notice, that the so-called full load characteristic line, i.e. operation points with the highest possible torque for a given revolution frequency, is given by valid data points, which

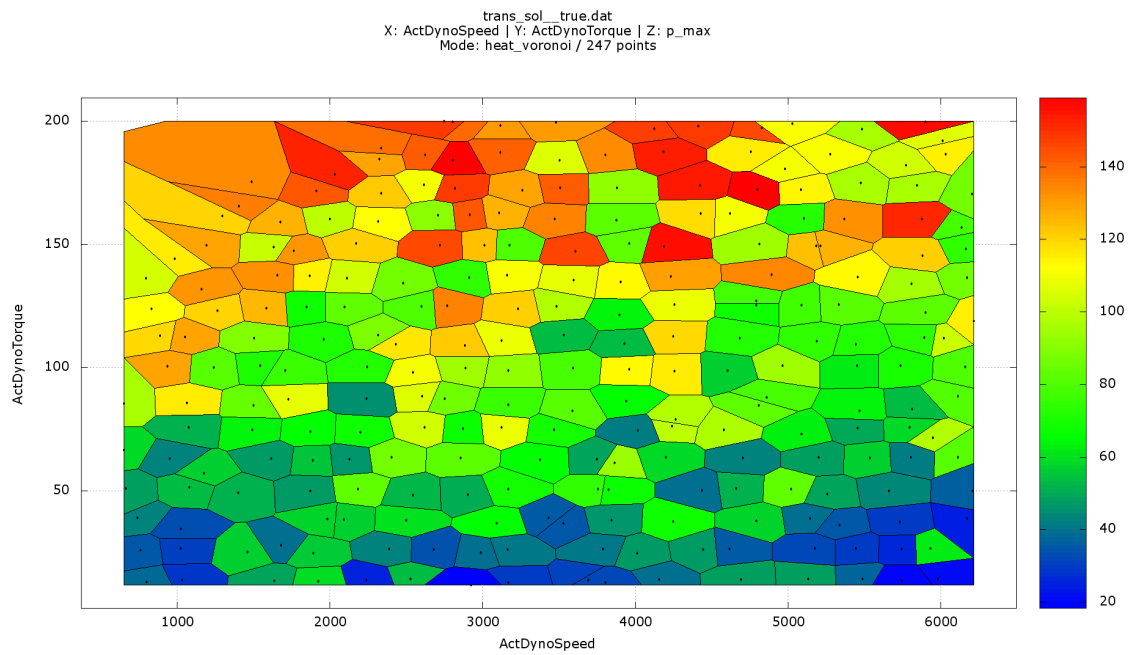




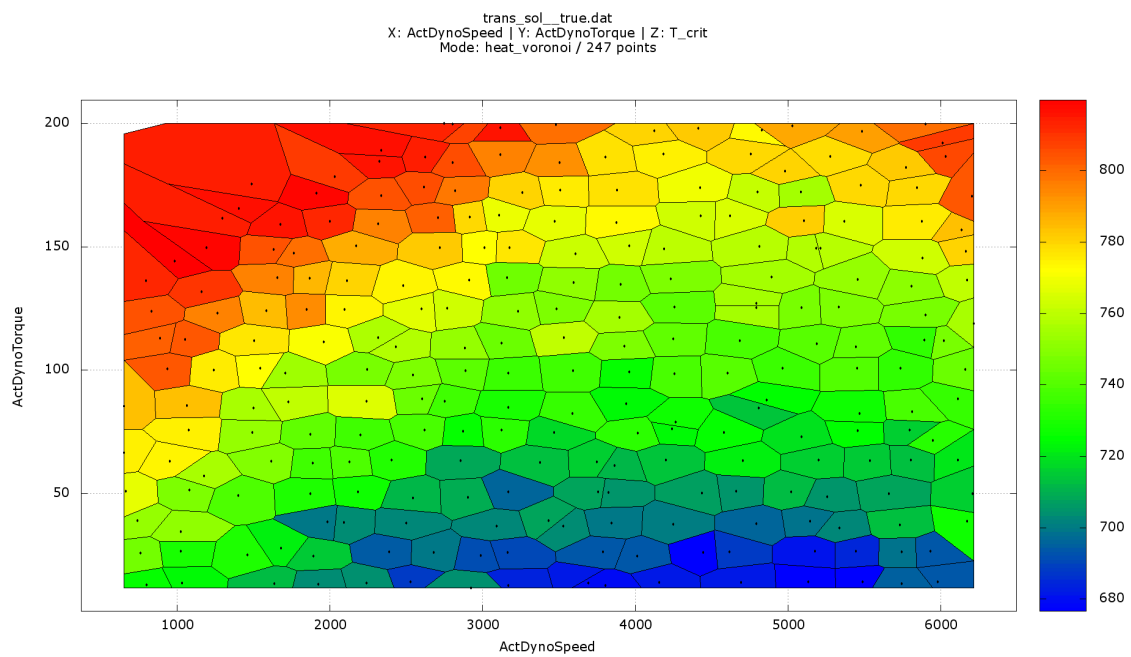
**Figure 113:** Data point distribution after stage 3 in VTB calibration

are close to the maximal critical temperature. That is why we do not observe any higher torque operation points. A similar, but not so clear picture of the same behavior is given in Figure 114. It shows the critical pressure, which is limited to 160bar.

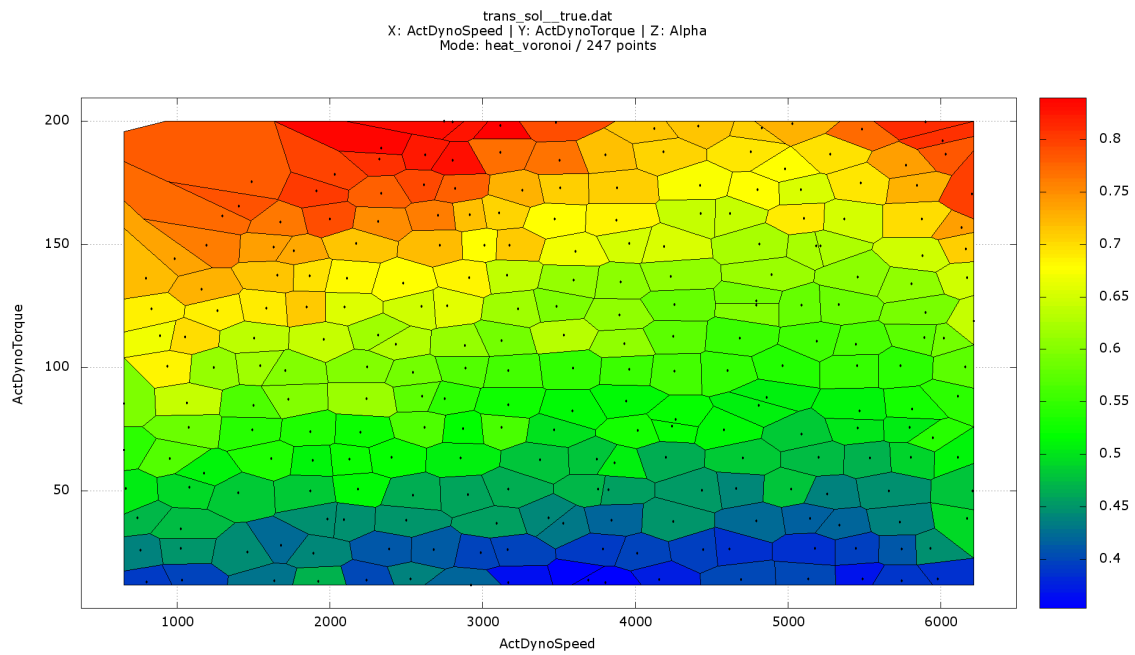
Figure 116 and Figure 117 show the successful settings of the transient solution for the *air valve* and *exhaust valve* actuator. In contrast to the previously derived setting for the *air valve* actuator with the *full factorial* method in Figure 109, we observe a smooth engine map that does not violate the tolerance constraint. The *exhaust valve* settings show the same systematic behavior, which have been observed with the full factorial method (see Figure 111), but Figure 117 illustrates the smooth transition from late opening angles for small revolution frequencies to early opening angles for higher revolution frequencies. Consequently, this solution satisfy the tolerance constraint.



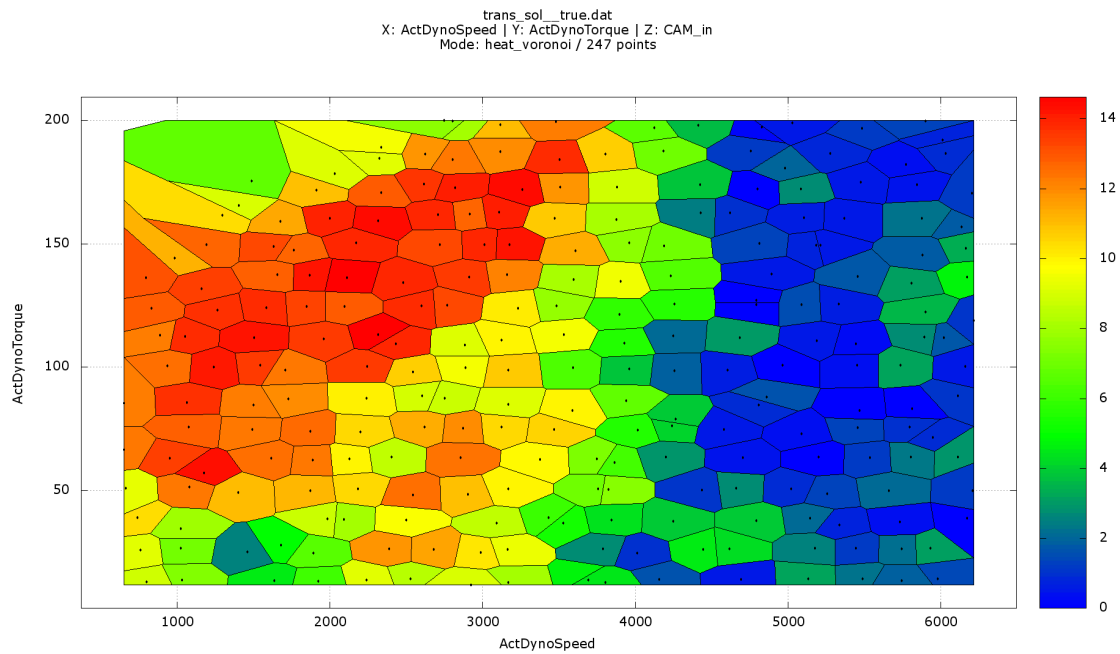
**Figure 114:** Resulting sensor values for critical inner cylinder pressure for transient solution in VTB calibration



**Figure 115:** Resulting sensor values for critical exhaust temperature for transient solution in VTB calibration



**Figure 116:** Setting of air valve angle for transient solution in VTB calibration



**Figure 117:** Setting for phase shift of exhaust camshaft for transient solution in VTB calibration

---

## 6 Summary and Outlook

---

This work focuses on an alternative approach in the field of advanced calibration methods. An algorithm is proposed for the calibration of internal combustion engines. This generic algorithm follows an autonomous measurement principle, which contains measurement planning, measurement path routing and measurement specialization.

This work introduces the new quasi-stationary measurement technique. So one saves measurement time, because one does not wait for stationary conditions before the measurement is recorded. Furthermore, this work discusses the new automatic refinement method, which additionally reduces the total measurement time. Both refinement criteria are presented: the refinement due to the derivation of new data points to a fit to a local second order polynomial function and the improvement of the unconditional stationary engine map. Data points for the local polynomial function are selected by the applied Voronoi tessellation algorithm of this work. Furthermore, this work introduces a new efficient way of temporal and spatial data compression. So, the algorithm is able to reduce the number of raw data points significantly, but without losing critical contents. Moreover, the algorithm is able to validate new data points by comparison of new points with nearest neighbor data points, which are selected by a Voronoi tessellation.

This work aims for a holistic mathematical approach that describes the optimization problem of the calibration processes. Due to the fact that this work expresses the problem as a linear integer program, one does not have to apply data synthetization, modeling or other kinds of data interpretations. So this algorithm is not subject to interpretation or systematical errors during the optimization process.

The proposed analytical solving process with IBM Cplex or SCIP deals with many variables, many criteria and many constraints. That is why the solver is able to determine the settings of all dynamic actuators of an engine, by maintaining hard and hard integral constraints for any operational point. The composition of the linear integer program is generic, so one is able to include additional constraints, if needed. The presented optimization is based on two emission test procedures, the New European Driving Cycle and the new Real-World Emissions Cycle RANDOM.

During this work, the impact of test procedures on the application results has been studied. It has been shown that in the current proposed structure of test procedures, the application results have low predictive power only.

The whole system is benchmarked, especially the comparison of the new method to full factorial measurements. This work shows that due to refinement techniques, the new algorithm is able to detect stationary optimal settings much faster than with a full factorial measurement. The measurement time at the engine test bench of 360 hours can be reduced to less than 40 hours while maintaining compliance to the same optimization quality.

Finally, the mode of operation and the optimization quality of the whole engine application process with the new introduced methods are presented. Therefore, one applies the new methods to two engine models, which are a substitute for a real engine test bench. The first model is based on a simulation of a spark-ignited engine, equipped with variable camshafts and inlet manifold injection. The second model is contributed by AVL List GmbH. This model has been created by measured data of an engine test bench. The diesel engine is equipped with a variable-geometry turbocharger and with a single pilot injection. Both models are applied like real engine benches, so both models are extended with a statistical and a systematical error.

---

This work discusses the complete optimization of both models. So, this work shows that the new algorithms perform as desired. Both models are optimized due to the specific fuel consumption. Furthermore, all maximal emission constraints and hard bounds are maintained. The maximal integral emission constraints are derived due to the new European driving cycle NEDC and the newly introduced real world Emission cycle RANDOM. Finally, the transient solution is derived by conserving all maximal gradients in the engine maps of each actuator.

This work proves, that the potential of optimization of modern internal combustion engines has not been exhausted. Particularly, the treatment of many criteria of the optimization shows that even smaller emission and fuel consumption values are possible. Especially, the direct creation of transient engine maps instead of stationary ones reduces the time effort of the application process in the road trial. In closing, this work investigates how test cycles, proposed by the European Union, are the crucial factors for the quality of the application process. Therefore, this work proposes the introduction of random based test cycles, in order to better reflect reality.

The future work relating to this project may include many further improvements. In order to reduce the measurement time, one should switch from the applied density-driven measurement approach to a profit-driven approach. Within this approach, the algorithm is able to estimate the possibility of the existence of a valuable parameter combination in a small region. In the case of a low probability, the measurement would be able to skip this region and save test bench time.

Another improvement focuses on the binary tree, which describes the data space. This work discussed the refinement methods, in order to adapt to the importance of specific regions in the data space. In order to reduce memory consumption and run-time it is necessary to coarsen parts of the binary tree.

Another topic on the list of further improvements are emission test procedures. As this work showed, the calibration result depends strongly on the emission cycles, so one has to define an emission cycle with a better predictive power of fuel consumption and emissions in the real world. So this work proposes a superposition of many randomly derived RANDOM cycles. Since these cycles are based on statistical data of real drivers, the superposition of many cycles will improve the reliability of the calibration results.

The last and biggest point of improvement is the implementation of this algorithm on a real test bench. This work showed that the algorithm maintains all functions that are necessary to measure, clean and optimize data. The algorithm runs already on a raw data signal with statistical and systematical error components, in order to test the applied cleaning and fitting methods. But the handling of a real test bench may be much more difficult.

So, one has to implement a bi-directional data interface from the test bench operating system to *AmmOC*. Furthermore, one has to adjust the cleaning methods to the real world data stream. Afterwards, one is able to handle the new measurement problem. The used test bench gradients and Solution field gradients are assumed: that is why one should measure theses major quantities. This has been demanded by P. Lind in [Lin12].

Another big topic is the test bench configuration. As described, a test bench includes a lot of controllers, which are treated as dynamic or static actuators by *AmmOC*. Some controllers are needed to maintain a stable operation of the internal combustion engine, but some are based on the stationary measurement method itself. So, finally, this work suggests the study of the appliance of controllers on dynamic actuators. Because this method applies a slow transient measurement principle it is possible to replace controllers, with direct actuators,

---

e.g. the revolution frequency is given by the power output of the engine, which is a result of all actuators and the configuration of the test bench braking system. Generally one should avoid to control dynamic actuators, which are in direct relationship to the 'air-path' of the engine, in order to reduce measurement time and in order to avoid controller oscillations.

---

## 7 Appendix

---

### 7.1 Quick Start Manual

---

#### 7.1.0.3 Installation Guide User Edition

In order to install the basic user edition of the "*Automatic multidimensional, multicriterial optimization algorithm for the calibration of internal combustion engines*" (*AmmOC*), one has to perform the following steps:

1. Install *ActiveState Perl x64*
  - a) Standard installation of *ActiveState Perl x64*
  - b) Install missing sources for *DATA\_Server/server.pl* with `> ppm install ...`
  - c) Finished!
2. Install *OpenCL*
  - a) Make sure, that your *CPU* supports OpenCL, i.e. Intel i5 or Intel i7
  - b) Make sure, that your *GPU* supports OpenCL, please use AMD-GPUs
  - c) Download *Intel SDK*: [Intel SDK 2013 x64 Installer](#)
  - d) Standard installation of Intel SDK
  - e) Download *AMD SDK*: [AMD SDK x64 Installer](#)
  - f) Standard installation of AMD SDK
  - g) Finished!
3. Install Python 2.7 x64
  - a) Download *ActiveState Python 2.7 x64*: [ActiveState Python 2.7 x64 Installer](#)
  - b) Standard installation of *ActiveState Python*
  - c) Missing sources may be downloaded as pre-builds from [Pre-Builds Repository](#)
  - d) Finished!
4. Install *PyOpenCL*
  - a) Download *PyOpenCL* for *Python 2.7 x64* from [Pre-Builds Repository](#)
  - b) Standard installation of *PyOpenCL*
  - c) Finished!
5. Install *PyQT4*
  - a) Download *PyQT4* installer: [PyQT4 installer](#)
  - b) Standard installation of *PyQT4*
  - c) Finished!
6. copy the directories *AmmOC* and *DATA\_Server* to your local SSD harddisk
7. Build the Qt forms
  - a) At command prompt: change to *AmmOC/building\_scripts*

- 
- b) At command prompt: Execute *build\_all.bat*
  8. Create initial configuration files (Templates for test bench setup at Elring Klinger)
    - a) Copy *config/EK\_DEMO\_config.conf* to *config/config.conf*
    - b) Copy *config/EK\_DEMO\_properties.prop* to *config/properties.prop*
    - c) Copy *config/EK\_DEMO\_INCA\_CAN\_Config.ICco* to *config/INCA\_CAN\_Config.ICco*
  9. Install *Integer Program* solver, i.e. *IBM Cplex* or *SCIP*
    - a) *IBM Cplex* adds an entry to the environment variable *PATH*. So, please make sure, that you can access *cplex.exe* from command prompt.
    - b) *SCIP* has to be copied to any location on your harddisk. Make sure, that it's path is added to the environment variable *PATH*
  10. Install *GnuPlot* and *GnuTamer*
    - a) *GnuTamer* has been tested with *GnuPlot 4.6*
    - b) Download *GnuPlot*: [GnuPlot 4.6 Installer](#)
    - c) Install *GnuPlot 4.6*
    - d) Installation of *GnuTamer* is supported by [M. Mertz](#)

#### 7.1.0.4 Installation Guide Developer Edition

In order to develop new routines for *AmmOC*, one has to install some additional tools:

1. Install *Boost* and *Python-Boost*
  - a) Download *Boost*: [Boost Source](#)
  - b) Configure *Boost* for *Python 2.7 x64*
  - c) Build *Boost*
2. Install *Tortoise SVN* and connect to repository
  - a) Download *Tortoise SVN*: [Tortoise SVN Installer](#)
  - b) Standard installation of *Tortoise SVN*
  - c) Checkout *AmmOC* repository *Repo* to local harddisk
3. Install Visual Studio 2012
  - a) Develop new x64 *Python* sources with Visual Studio 2012. Examples can be found in *src/inc\_cpp/Greedy\_Ants*

#### 7.1.0.5 Start *AmmOC*

1. At command prompt: Change to *DATA\_Server* directory
2. At command prompt: Start TCP/IP-Server with *server.pl*
3. If everything has been installed correctly, it should appear *>Connecting to: test\_db.sqlite*
4. Open a second command prompt
5. At command prompt: Change to *AmmOC* directory



6. Launch *AmmOC* graphical user interface with *b\_AmmOC\_GUI.py*
7. Arrange *AmmOC* GUI to the right with *Windows* button + <Right>
8. Arrange *AmmOC* Log to the left with *Windows* button + <Left> (see Figure 118)

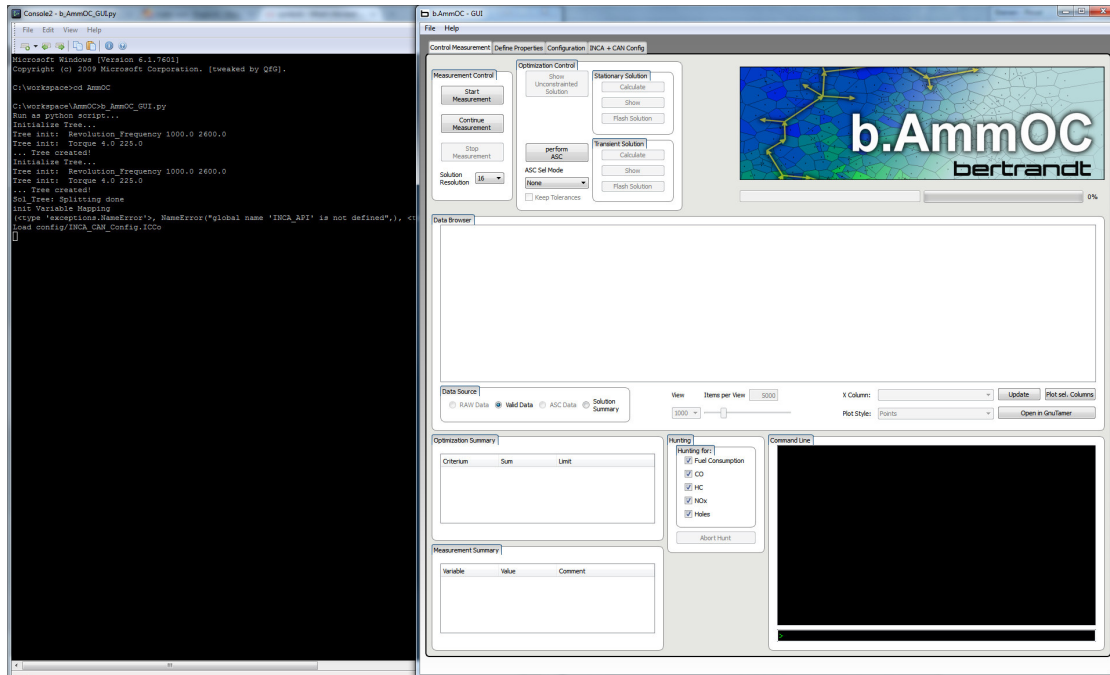


Figure 118: Successful start of *AmmOC*

#### 7.1.0.6 Parameter configuration

The parameter configuration defines all mathematical variables for the optimization of the internal combustion engine. Each parameter of the optimization is configured on the slide *Define Properties* (see Figure 119).

In case of a new combustion engine, which has to be optimized, all three rows should be empty or cleared by the user. In certain cases, the user may open an existing properties (\*.prop) file of a similar engine. Afterwards, the user has to specify it's optimization problem.

The first row *dynamic Actuators* describes the set of variable actuators. These actuators are varied in order to optimize the combustion engine. By pressing the + button at the end of the row, the user can add an additional entry. An existing entry may be erased by pressing the - button. During the creation of a new entry, the user specifies the type, e.g. *dyn\_Act*, *stat\_Act* and *result*, of the new parameter. The parameters is sort into the corresponding row. Furthermore, the user has to define the upper and lower border (*low/high limit*) of the new parameter. In case of actuators, the program defines measurement processes within the given borders of the dynamic actuators. In case of *results*, the measurement ramp is interrupted if any sensor value exceeds the given borders and it's *criteria type* is *hard*, *soft* or *hard\_integral*.

The *bench gradient* defines the absolute adjustment speed per second, in other words, it defines the maximal slope of each measurement relating to the corresponding dimension. The *default value* is the initial value of each actuator. Since *static actuators* are not modified,



---

they remain on its *default value*. The boolean variable *solution field* represents the affiliation of the parameter to the solution map. Typically, the solution map is given by the engine revolution frequency and the engine torque or torque-relevant injection mass. *Column* is generated automatically and describes the column number of the parameter in the generated ASCII output. If a user wants to apply an additional engine model, besides *VTB* and *AVL\_MM*, the user has to modify the function *start\_TB* in *AmmOC/b\_AmmOC.py*, so that it generates a list of list with the given sequence (*columns*) of result values. The last parameter *solution gradient* defines the maximal relative slope in the generated transient solution maps.

**Example::** The *solution gradient* of *injected\_fuel\_quantity* is 10. Consequently, this parameter may be varied ten-times from min to max, if a single *solution field* parameter, i.e. *revolution\_frequency* is varied from min to max. If one varies two *solution field* parameters, which should be the standard case, the maximal variation of the *injected\_fuel\_quantity* increased to ca. 14.

In the last row, the user defines the properties of the sensor variables. As already mentioned, the user defines the upper and lower borders and the *criteria type* of the sensor. If the *criteria type* is equal to *ignore*, the measurement process will ignore the borders of this sensor, but the borders of the sensor variable is the foundation of the digitalization of its values on the CAN-bus. The *upper\_fit\_limit* is important relating to the local refinement model and defines the maximal allowed deviation of the newly measured sensor value to the local model. If it exceeds the maximal deviation and the *local\_polynomial\_model\_refinement* is active, the algorithm refines the data space structure in this region. Finally, *hard\_integral\_limit* defines the maximal integral value of the sensor value with respect to the corresponding emission cycle (*driving\_cycle*).

#### 7.1.0.7 General configuration

On the slide *configuration*, the user may define general properties of the optimization process (see Figure 120). Each configuration field is explained in the graphical user interface in a custom tool tip.

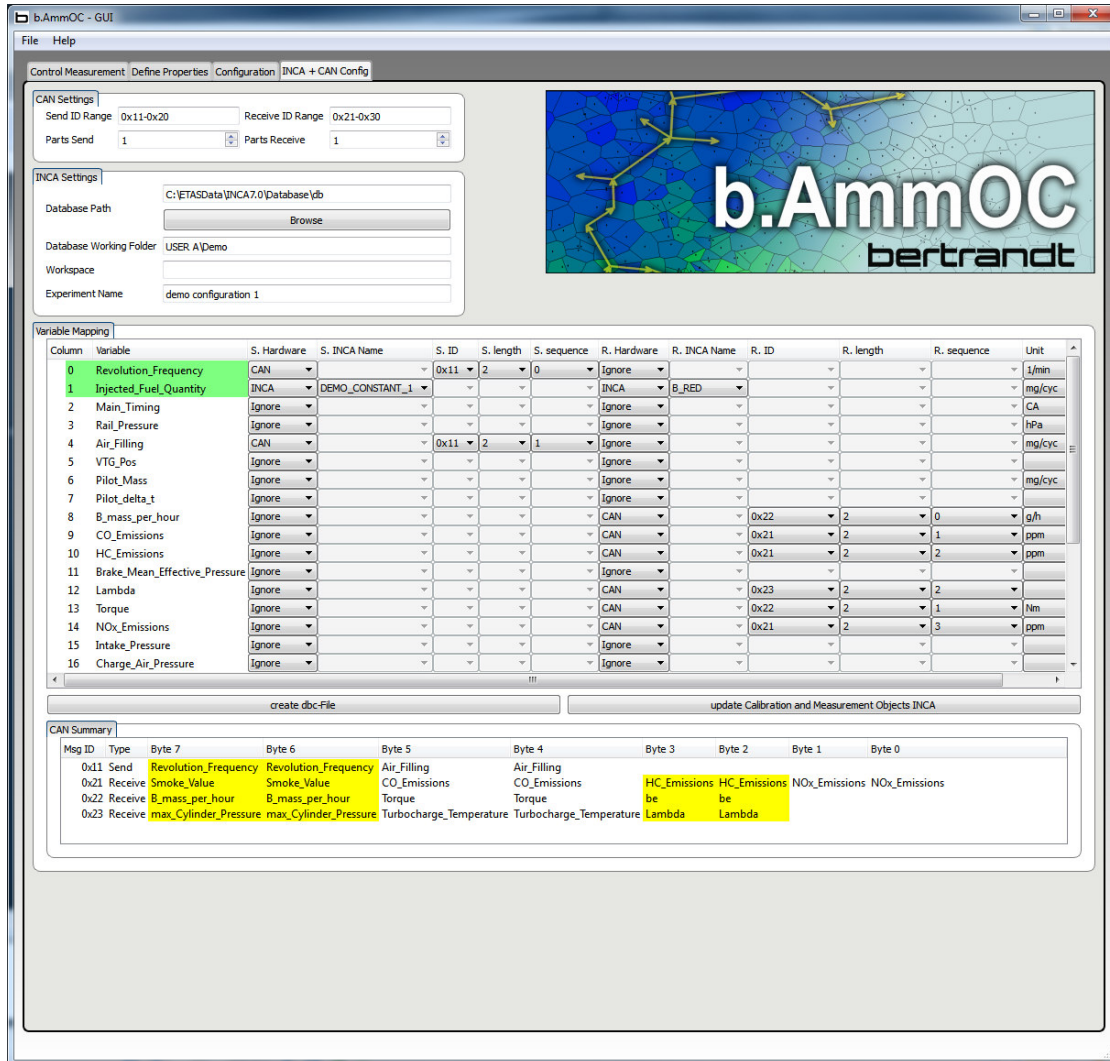
#### 7.1.0.8 Test bench interface

On the slide *INCA + CAN Config*, the user configures the test bench interface (see Figure 121). In the group box *CAN settings*, one defines the available ID-range for Send and Receive. Furthermore, the user defines the split between sending and receiving on the CAN-bus. If *Parts Send* is 9 and *Parts Receive* is equal to 1, the system transmits once but receives 9-times. With a measurement frequency of 10Hz, the program would send each second and receive nine-times per second. The group box *INCA settings* holds all parameters of one specific experiment in ETAS-INCA.

In the big group box *Variable Mapping*, the user has to map each inner variable (*Variable*) to its specific hardware settings. The first five columns after *Variable* define the sending attributes, so the inner variable is transmitted to the test bench. The last five columns define, how these sensor values are received. Typically, the most important sensor values are measured by the test bench and transmitted to *AmmOC*, but also the resulting actuator settings are measured, too and re-transmitted to *AmmOC*. So, the user can be sure, the actuators follow the defined actuator settings. The user may map an inner variable to the CAN-Bus, consequently, the user has to define the length of the digitalization, the ID of the message and the sequence within the message. *AmmOC* determines the necessary conversion factors and composes the messages on the CAN-Bus. In order to configure, the test bench system



**Figure 120:** General configuration of AmmOC



**Figure 121:** Test bench interface configuration of AmmOC

with the same settings than *AmmOC*, the user has to press *create dbc-File*. This standard file format contains the CAN-Configuration of the whole system. If the user maps the inner variable to an *INCA* variable, *AmmOC* modifies application label, curves and maps. In case of application curves and maps, *AmmOC* analyzes the dependencies of these application objects and modifies the curve or map only at the corresponding location.

## 7.2 Configuration of the reference problems

### 7.2.1 Simulation Model VTB

#### 7.2.1.1 Properties and configuration

**Table 8:** Actuator configuration of the *Virtual Test Bench* model

Actuator	column	default	grad	high	low	Solution_field
ActDynoSpeed [1/min]	0	2000	30	6250	650	true
CAM_ex [°CA]	1	0	0.1	11	-11	false
CAM_in [°CA]	2	0	0.1	15	-15	false
ZW [°CA]	3	340	0.4	360	300	false
AGR [%]	4	0.1	0.1	1	0	false
Alpha [°]	5	1	0.001	1	0.25	false

**Table 9:** Result configuration of the *Virtual Test Bench* model

Result	column	Criteria_type	hard_integral_limit	high	low	Solution_field
ActDynoTorque [Nm]	9	hard		200	2	true
b_e [g/kWh]	6	soft		1600	100	false
p_max [bar]	8	hard		160	0	false
T_crit [K]	7	hard		850	0	false

**Properties:**

**Configuration:**

**Table 10:** General configuration of the *Virtual Test Bench* model

Setting	Value
ASC__general_scale	5
GENERAL__dim	6
GENERAL__Gnuplot_path	E:/gnuplot/bin
GENERAL__IP_Server	ru-nx-104
GENERAL__Port_Server	29876
GENERAL__split_depth	10
GENERAL__VTB_Mode	VTB
SPECIALIZATION__split_limit	0.5
TESTDRIVE__cycles	5000
TESTDRIVE__homogenous_draws	1
TESTDRIVE__meas_frequency	1
TESTDRIVE__measurement_time	10000
TESTDRIVE__method	dyn_Testdrive
TESTDRIVE__mode	diagonal
TSC__time_series_lowest_measurement_frequency	0.1
TSC__time_series_percentage_change	0.03

## 7.2.2 DoE Model AVL

### 7.2.2.1 Properties and configuration

**Table 11:** Actuator configuration of the *AVL DoE* model

Actuator	column	default	grad	high	low	Solution_field
Revolution_Frequency [1/min]	0	1500	1	2600	1000	true
Injected_Fuel_Quantity [ $mm^3$ /cyc]	1	20	0.1	60	6	false
Main_Timing [ $^{\circ}$ CA]	2	3	0.2	10	0	false
Rail_Pressure [hPa]	3	300000	100	1126537	295677	false
Air_Filling [mg/hub]	4	300	5	991	275	false
VTG_Pos [%]	5	40	1	85	30	false
Pilot_Mass [ $mm^3$ /inj]	6	1	0.1	1.5	0.4	false
Pilot_delta_t [ $\mu$ s]	7	2000	10	2565	1540	false

**Properties:**

**Configuration:**

**Table 12: Result configuration of the AVL DoE model**

Result	column	Criteria_type	hard_integral_limit	high	low	Solution_fiel
B_mass_per_hour [kg/h]	8	ignore		100	0	false
CO_Emissions [ppm]	9	ignore		100000	0	false
HC_Emissions [ppm]	10	ignore		100000	0	false
Brake mean effective pressure [bar]	11	ignore		100	0	false
Lambda []	12	ignore		20	0	false
Torque [Nm]	13	ignore		300	0	true
NOx_Emissions [ppm]	14	ignore		100000	0	false
Intake_Pressure [bar]	15	ignore		5	0	false
Charge_Air_Pressure [bar]	16	ignore		10	0	false
max_Cylinder_Pressure [bar]	17	ignore		150	0	false
Smoke_Value [fsn]	18	ignore		1000	0	false
Intake_Temperature [°C]	19	ignore		500	0	false
Turbocharge_Temperature [°C]	20	hard		1200	0	false
CO_Emissions_per_hour [g/h]	21	hard integral	9.253 (*3600)	3600	0	false
HC_Emissions_per_hour [g/h]	22	hard integral	4.2564 (*3600)	3600	0	false
NOx_Emissions_per_hour [g/h]	23	hard integral	3.3311 (*3600)	3600	0	false
be [g/kWh]	24	soft		1000	0	false

**Table 13: General config of AVL DoE model run**

Setting	Value
ASC__general_scale	5
GENERAL__dim	8
GENERAL__Gnuplot_path	E:/gnuplot/bin
GENERAL__IP_Server	ru-nx-104
GENERAL__Port_Server	29876
GENERAL__split_depth	10
GENERAL__VTB_Mode	AVL_MM
SPECIALIZATION__split_limit	0.5
TESTDRIVE__cycles	5000
TESTDRIVE__homogenous_draws	1
TESTDRIVE__meas_frequency	1
TESTDRIVE__measurement_time	10000
TESTDRIVE__method	dyn_Testdrive
TESTDRIVE__mode	diagonal
TSC__time_series_lowest_measurement_frequency	0.1
TSC__time_series_percentage_change	0.03



---

## References

---

- [Ata07] K Atashkari, N Nariman-Zadeh, M Golcu; '*Modelling and multi-objective optimization of a variable valve-timing spark-ignition engine using polynomial neural networks and evolutionary algorithms energy conversion*'; **Energy Conversion and Management**, Volume 48, Issue 3, March 2007, Pages 1029-1041; [2007]
- [Art04] M. Adreé; '*Real-world driving cycles for measuring cars pollutant emissions - Part A*'; **INRETS, Institute national de recherche sur les transports et leur securite**; [2004]
- [AVL13] M. Vogels et al.; '*DoE Model "Compression ignition engine"*'; **AVL LIST GmbH; Hans-List-Platz 1; A 8020 Graz; Austria**; [2013]
- [Bar96] C. Barber, D. Dobkin, H. Huhdanpaa; '*The Quickhull Algorithm for Convex Hulls*'; **ACM Transactions on Mathematical Software**, Vol. 22, No. 4; [1996]
- [Bas07] D. Basak, S. Pal, D. Patranabis; '*Support Vector Regression*'; **Neural Information Processing - Letters and Reviews**, Vol. 11, No. 10; [2007]
- [BMW03] K. Knödler, T. Fleischhauer, A. Mitterer, S. Ullmann, A. Zell; '*Modellbasierte Online-Optimierung moderner Verbrennungsmotoren Teil 2 Grenzen des fahrbaren Suchraums*'; **ATZ Edition: 2003-06**; [2003]
- [BMW3a] J. Poland, K. Knödler, T. Fleischhauer, A. Mitterer, S. Ullmann, A. Zell; '*Modellbasierte Online-Optimierung moderner Verbrennungsmotoren Teil 1 Aktives Lernen*'; **ATZ Edition: 2003-05**; [2003]
- [Bei10] Beidl, Bier; '*Engine-in-the-Loop - Effiziente Betriebsstrategien für den Verbrennungsmotor im Hybridantrieb*'; **AVL Tech Day Hybrid-Testing. Darmstadt**; [2010]
- [Ber02] D. Bertsimas and M. Sim; '*Robust Discrete Optimization and Network Flows*'; **MIT**; [2002]
- [Boi10] J.D. Boissonnat; '*Convex Hulls, Voronoi Diagrams and Delaunay Triangulations*'; **Winter School on Algorithmic Geometry; ENS-Lyon**; [2010]
- [Bor10] R. Borndörfer, M. Dür, A. Martin, S. Ulbrich; '*Optimierung I, Einführung in die Optimierung*'; **TU Darmstadt, Wintersemester 2010/2011**; [2010]
- [Bot11] S. Bott; '*personal correspondence*'; ; [2011]
- [Bro87] Bronstein, Semendjajew; '*Taschenbuch der Mathematik*'; **Verlag Harri Deutsch Thun und Frankfurt/Main**; [1987]
- [Buc12] C. Buchheim; '*Diskrete Optimierung*'; **TU Dortmund**; [2012]
- [Cas08] M. Cassola; '*Evolutive Algorithmen*'; **Universität Karlsruhe**; [2008]
- [Chw10] A. Chwatal; '*Solving the minimum label spanning tree problem by ant colony optimization*'; **Proceedings of GEM 2010, Las Vegas, Nevada**; [2010]
- [Dar59] C. Darwin; '*On the Origin of Species*'; <http://www.gutenberg.org/etext/1228>; [1859]

- 
- 
- [Dor00] A. Dorigo, E. Bonabeau, G. Theraulaz; '*Ant algorithms and stigmergy*'; **Future Generation Computer Systems** 16 (2000) 851-871; [2000]
- [Die07] DiePresse.com; 'Weltweit 900 Millionen Pkw unterwegs'; <http://diepresse.com/home/panorama/welt/315121/Weltweit-900-Millionen-Pkw-unterwegs->; [2007]
- [EC07] European Commission; '*Regulation (EC) No 715/2007 of the European Parliament and of the Council of 20 June 2007 on type approval of motor vehicles with respect to emissions from light passenger and commercial vehicles (Euro 5 and Euro 6) and on access to vehicle repair and maintenance information*'; **EC - European Commission, Official Journal of the European Union L171**; [2007]
- [EC08] European Commission; '*Regulation (EC) No 692/2008 of 18 July 2008 implementing and amending Regulation (EC) No 715/2007 of the European Parliament and of the Council on type-approval of motor vehicles with respect to emissions from light passenger and commercial vehicles (Euro 5 and Euro 6) and on access to vehicle repair and maintenance information*'; **EC - European Commission, Official Journal of the European Union L199**; [2008]
- [EC09] European Commission; '*Regulation (EC) No. 443/2009 of the European Parliament and of the Council of 23 April 2009 setting emission performance standards for new passenger cars as part of the Community's integrated approach to reduce CO2 emissions from light-duty vehicles.*'; **EC - European Commission, Official Journal of the European Union L 140**; [2009]
- [EEA09] EEA - European Environmental Agency; '*Exceedance of air quality limit values in urban areas (CSI 004) - Assessment published Dec 2009*'; <http://www.eea.europa.eu/dataand-maps/indicators/exceedance-of-air-quality-limit-1/exceedance-of-air-quality-limit-1>'; [2009]
- [EEA10] EEA - European Environmental Agency; '*European Union emission inventory report 1990 - 2008 under the UNECE convention on Long-range Transboundary Air Pollution (LRTAP)*'; '**EEA Technical report No 7/2010**'; [2010]
- [Ede87] H. Edelsbrunner; '*Algorithms in Combinatorial Geometry*'; **Springer-Verlag**; [1987]
- [Epp02a] D. Eppstein; '*Priority dictionary using binary heaps*'; **UC Irvine**; [2002]
- [Epp02b] D. Eppstein; '*Dijkstra's algorithm for shortest paths*'; **UC Irvine**, <http://aspn.activestate.com/ASPN/Cookbook/Python/Recipe/117228>; [2002]
- [Eri12] J. Erickson; '*Algorithms*'; **University of Illinois at Urbana-Champaign**; [2012]
- [Fuk04] K. Fukuda; '*Frequently Asked Questions in Polyhedral Computation*'; **Swiss Federal Institute of Technology**; [2004]
- [Gal12] J. Gallier; '*Notes on Convex Sets, Polytopes, Polyhedra, Combinatorial Topology, Voronoi Diagrams and Delaunay Triangulations*'; **Department of Computer and Information Science; University of Pennsylvania**; [2012]

- 
- 
- [Geb11] M. Gebhard; *'Entwicklung eines Deterministik Walker Algorithmus zur optimalen Kombination von Teiloptima zu einem globalen Optimum'*; **Hochschule München, Informatik und Mathematik (not published)**; [2011]
- [Gol05] R. Golloch; *'Downsizing bei Verbrennungsmotoren'*; **Springer**; [2005]
- [Gro10] H. Grohe, G. Russ; *'Otto- und Dieselmotoren'*; **Vogel Business Media**; [2010]
- [Gro11] K. Grote; *'Dubbel Taschenbuch für den Maschinenbau'*; **Auflage 23, Springer**; [2011]
- [Gsc01] K. Gschweidl; *'Steigerung der Effizienz in der modellbasierten Motorenapplikation durch die neue CAMEO Online DoE-Toolbox'*; **ATZ - Automobiltechnische Zeitschrift, July 2001, Volume 103, Issue 7-8, pp 636-643**; [2001]
- [Hee05] D. Heermann; *'Vorlesung Statistische Mechanik: Random Walk'*; **Univeristät Heidelberg**; [2005]
- [Hli12] P. Hlineny, O. Moris; *'Dynamic Scope-Based Dijkstra's Algorithm\*'*; **Faculty of Informatics, Masaryk University**; [2012]
- [Hir02] T. Hiroyasu, M. Miki, J. Kamiura, S. Watanabe, H. Hiroyasu; *'Multi-Objective Optimization of Diesel Engine Emissions and Fuel Economy using Genetic Algorithms and Phenomenological Model'*; **Doshisha University, 02FFL-183**; [2002]
- [IAV02] G. Buschmann, K. Röpke: IAV; *'Optimierung der Motorenentwicklung durch neue Wege in der Entwicklungssystematik'*; ; [2002]
- [Ise00] R. Isermann; *'Einsatz schneller neuronaler Netze zur modellbasierten Optimierung von Verbrennungsmotoren Teil1 und Teil2'*; **MTZ - November 2000, Volume 61, Issue 61**; [2000]
- [Ise03] R. Isermann; *'Effiziente Motorapplikation mit lokal linearen neuronalen Netzen'*; **ATZ Edition 2003-05**; [2003]
- [Ise10] R. Isermann; *'Elektronisches Management motorischer Fahrzeugantriebe'*; **ATZ/MTZ-Fachbuch, Vieweg-Teubner**; [2010]
- [Ise05b] Isermann, Zimmerschied; *'Stationäre und dynamische Motorvermessung zur Auslegung von Steuerkennfeldern - Eine kurze Übersicht'*; **Automatisierungstechnik 2005 53:2-2005, 87-94**; [2005]
- [Jos02] M. Joswig; *'Beneath-and-Beyond Revisited'*; **TU Berlin, Institut für Mathematik**; [2002]
- [JRC13] M. Weiss, P. Bonnel, R. Hummel, N. Steininger; *'A complementary emissions test for light-duty vehicles: Assessing the technical feasibility of candidate procedures'*; **Report EUR 25572 EN; European Commission**; [2013]
- [JRC13a] M. Weiss, P. Bonnel, R. Hummel, U. Manfredi, R. Colombo, G. Lanappe, P. Li-jour, M. Sculati; *'Analyzing on-road emissions of light-duty vehicles with Portable Emission Measurement System (PEMS)'*; **Report EUR 24697 EN; European Commission**; [2011]

- 
- [Kar99] O. Karch; '*Algorithmische Geometrie*'; **Lehrstuhl für Informatik I**; [1999]
- [Kas11] A. Kasperski; '*DISCRETE OPTIMIZATION AND NETWORK FLOWS*'; **Wroclaw University of Technology**; [2011]
- [Klo10] A. Klöckner; '*Scripting GPUs with PyOpenCL*'; **Division of applied mathematics, Brown University**; [2010]
- [Koe06] E. Köhler and R. Flierl; '*Verbrennungsmotoren*'; **vieweg**; [2006]
- [Kon09] T. Konstantopoulos; '*MARKOV CHAINS AND RANDOM WALKS*'; **Uppsala University**; [2009]
- [Kor08] B. Korte and J. Vygen; '*Kombinatorische Optimierung*'; **Springer**; [2008]
- [Kru09] T. Kruse, S. Kurz; '*Modern Statistical Modeling and Evolutionary Optimization Methods for the Broad Use in ECU Calibration*'; **ETAS GmbH, P.O. Box 30 02 20, 70442 Stuttgart**; [2009]
- [Kru11] S. Krumke; '*Data Structures and Algorithms for Combinatorial Optimization WS 11/12*'; **TU Kaiserslautern**; [2011]
- [Lin12] P. Lind; '*Entwicklung zeitbasierter dynamischer Kennfelder für die transiente Optimierung von Verbrennungskraftmaschinen*'; **TU Darmstadt**; [2012]
- [Low98] D. Lowe, K. Zapart; '*Point-wise Confidence Interval Estimation by Neural Networks: A comparative study based on automotive engine calibration*'; **Aston University**; [1998]
- [Lue02] G. Lütkemeyer; '*Effektive Strategien für Motorsteuerungsapplikationen*'; **MTZ - Motortechnische Zeitschrift, July 2002, Volume 63, Issue 7-8, pp 602-611**; [2002]
- [Mal08] A. Malikopoulos, D. Assanis, and P. Papalambros; '*Optimal Engine Calibration for Individual Driving Styles*'; **Automotive Research Center, The University of Michigan**; [2008]
- [Mar02] H. Marchand, A. Martin, R. Weismantel, L. Wolsey; '*Cutting planes in integer and mixed integer programming*'; **Elsevier; Discrete Applied Mathematics; Volume 123, Issues 1-3**; [2002]
- [Mar03] F. Markowetz; '*Klassifikation mit Support Vector Machines*'; **Max-Planck-Institut für Molekulare Genetik**; [2003]
- [Mer10] M. Mertz; '*Programmierung und Validierung eines mehrdimensionalen Optimierungsalgorithmus für Verbrennungsmotoren*'; **Hochschule RheinMain, Fachbereich Design Informatik Medien**; [2010]
- [Mir11] A. Mirzaian; '*Convex Hull in 3D & Higher Dimensions*'; **COSC 6114**; [2011]
- [Mit00] A. Mitterer; '*Optimierung vielparametrischer Systeme in der KFZ-Antriebsentwicklung*'; **VDI-Verlag**; [2000]
- [Mit0a] A. Mitterer; '*Modellgestützte Kennfeldoptimierung – Ein neuer Ansatz zur Steigerung der Effizienz in der Steuergeräteapplikation*'; **TODO**; [2000]

- 
- [Mit0b] A. Mitterer; '*Einsatz von Softcomputing-Techniken zur Kennfeldoptimierung elektronischer Motorsteuergeräte*'; **Automatisierungstechnik** - 48; [2000]
- [Mun12] A. Munshi, B. Gaster, T. Mattson, J. Fung, D. Ginsburg; '*OpenCL - Programming Guide*'; **Addison-Wesley**; [2012]
- [Nie09] T. Nieberg; '*Lineare und Ganzzahlige Optimierung*'; **Universität Bonn**; [2009]
- [Ott99] T. Ottmann; '*Algorithmische Geometrie*'; **Universität Freiburg**; [2000]
- [Pea05] K. Pearson; '*The Problem of the Random Walk*'; **Nature**. 72, 294; [1905]
- [Pel06] L. Pelkmans, P. Debal; '*Comparison of on-road emissions with emissions measured on chassis dynamometer test cycles*'; **Transportation Research Part D** 11; [2006]
- [Pfe12] M. Pfetsch; '*Diskrete Optimierung*'; **TU Darmstadt, Sommersemester 2012**; [2012]
- [Pól21] G. Pólya; '*Über eine Aufgabe der Wahrscheinlichkeitsrechnung betreffend die Irrfahrt im Straßennetz*'; **Mathematische Annalen**. 84, Nr. 1-2; [1921]
- [Ras04] E. Rask and M. Sellnau; '*Simulation-Based Engine Calibration: Tools, Techniques, and Applications*'; **Delphi Research Labs**; [2004]
- [RFH07] RFH; '*Otto - und Dieselmotoren*'; **SAIYA.DE**; [2007]
- [Rig11] G. Righini; '*Discrete optimization problems Complements of Operations Research*'; **Universita degli Studi di Milano**; [2011]
- [Roe07] K. Röpke; '*Rapid Measurement - Grundbedatung eines Verbrennungsmotors innerhalb eines Tages*'; **ATZ** 2007-04; [2007]
- [Sac09] L. Sachs, J. Hedderich; '*Angewandte Statistik, Methodensammlung mit R*'; **Springer**; [2009]
- [Sch06] G. Schreiber; '*Untersuchung von Verbesserungspotentialen hinsichtlich Verbrauch und Drehmoment bei Ottomotoren mit Hilfe 1-dimensionaler Simulationsrechnungen*'; **Maschinenbau und Verfahrenstechnik TU Kaiserslautern**; [2006]
- [Sch07] A. Schlosser; '*Beschleunigte Antriebsstrangentwicklung mittels modellbasierter Applikation*'; **MTZ - Motortechnische Zeitschrift**, February 2007, Volume 68, Issue 2, pp 134-141; [2007]
- [Sch08] C. Schmell; '*Entwicklung einer Methodik zur Integration der Variationsraumvermessung in die globale dynamische Motorvermessung*'; **Darmstadt, Technische Univ., Studienarbeit**; [2008]
- [Sch09] A. Schmitt, C. Beidl, B. Lenzen; '*Versuchsmethodik zur Entwicklung von Emissionsminderungsstrategien für Heavy-Duty Motoren*'; **MTZ-Konferenz Heavy Duty-, On- und Off-Highway-Motoren, Friedrichshafen**; [2009]
- [Sit06] S. Sitthiracha; '*AN ANALYTICAL MODEL OF SPARK IGNITION ENGINE FOR PERFORMANCE PREDICTION*'; **KING MONGKUT'S INSTITUTE OF TECHNOLOGY NORTH BANGKOK**; [2006]

- 
- [Sou07] A. Soung, A. Mitterer; '*Modellbasierte Online- Optimierung in der Simulation und am Motorenprüfstand*'; **MTZ 01/2007**; [2007]
- [Spi01] F. Spitzer; '*Principles of Random Walk*'; **Springer**; [2001]
- [Sri95] N. Srinivas, K. Deb; '*Muiltiobjective Optimization Using Nondominated Sorting in Genetic Algorithms*'; **MIT, Evolutionary Computation 2(3): 22 1-248**; [1995]
- [Ste02] R. Steinbrink; '*Verbesserung des Kraftstoffverbrauchs von Dieselmotoren Teil1 und Teil2*'; **MTZ - December 2002, Volume 63, Issue 12**; [2002]
- [Vap74] V. Vapnik and A. Chervonenkis; '*Theory of Pattern Recognition*'; **Nauka**; [1974]
- [Wei09] T. Weise; '*Global Optimization Algorithms - Theory and Application -*'; **<http://www.it-weise.de/>**; [2009]
- [Wei13] M. Weiss, P. Bonnel, R. Hummel, N. Steininger; '*A complementary emissions test for light-duty vehicles: Assessing the technical feasibility of candidate procedures*'; **EU Commission, EUR 25572 EN**; [2013]
- [Woe00] W. Woess; '*Irrfahrten*'; **TU Graz**; [2000]
- [Wor13] R. Löffler; '*Worldwide-data*'; **<http://www.worldwide-datas.com/autos/>**; [2013]
- [Yas08] S. Yaseen, N. AL-Slamy; '*Ant Colony Optimization*'; **IJCSNS International Journal of Computer Science and Network Security, VOL.8 No.6, June 2008**; [2008]



---

# LEBENS LAUF

---

## AUSBILDUNGSDATEN

---

JANUAR 2011 BIS  
FEBRUAR 2015

**Industriepromotionsarbeit im Bereich der angewandten Mathematik**

Graduiertenschule „Computational Engineering“ der TU Darmstadt  
Titel der Arbeit: „*Development of an automatic, multidimensional, multicriterial optimization algorithm for the calibration of internal combustion engines*“.

Verteidigung der Arbeit: 3. Februar 2015

---

30.11.2007

---

**Abschluss Physikstudium mit Diplom** (Note 1,2)

---

24.02.2005

---

Abschluss Vordiplom (Note 2,1)

---

APRIL 2003 BIS  
DEZEMBER 2007

---

**Diplom-Studium der Physik** mit dem Schwerpunkt auf Kernphysik und Computernetzwerkrechnungen an der Justus-Liebig-Universität Gießen

---

## BISHERIGE BERUFLICHE TÄTIGKEITEN

---

MAI 2014 BIS HEUTE

Projektleiter für die Firma **IAV GmbH** im Bereich Überwachungskonzept Steuergeräte nach AK EGAS

---

APRIL 2011 BIS  
DEZEMBER 2013

---

Lead Engineer in der Firma **Bertrandt AG** im Bereich der Software-Entwicklung und Vorentwicklung für Kalibriermethodik

---

SEPTEMBER 2008 BIS  
APRIL 2011

---

Entwicklungsingenieur in der Firma **Bertrandt AG** im Bereich der Vorentwicklung für Kalibriermethodik an Motorenprüfständen der **Adam Opel GmbH**

---

APRIL 2009 BIS  
APRIL 2010

---

Arbeitnehmerüberlassung durch die Firma **Bertrandt AG** an die Firma **Areva NP** in Offenbach im Bereich der Radiologie und Störfallsimulation in Kernkraftwerken

---

## FORSCHUNGSTÄTIGKEIT

---

JANUAR 2012 BIS OKTOBER 2012	<b>Betreuung der Masterarbeit</b> „Entwicklung zeitbasierter dynamischer Kennfelder für die transiente Optimierung von Verbrennungskraftmaschinen“
JULI 2009 BIS APRIL 2010	<b>Betreuung der Diplomarbeit</b> „Automatisierte mehrdimensionale multi-kritikale Optimierung von Motorenprüfständen durch Support Vector Regression und Deterministic Walker“
JULI 2009 BIS JANUAR 2010	<b>Betreuung der Diplomarbeit</b> „Automatisierte mehrdimensionale Optimierung von Motorprüfstandsdaten durch einen Random Walker“
SEPTEMBER 2008 BIS APRIL 2009	<b>Betreuung der Diplomarbeit</b> „Prozesstemperaturreduktion am Verbrennungsmotor mit Hilfe der Simulation durch GT-Power“
21.12.2007 BIS 31.10.2008	Wissenschaftlicher Mitarbeiter im II. Physikalischen Institut der Justus-Liebig-Universität Gießen
17.12.2007 BIS 18.12.2007	Experiment zur Speicherung von Ionen im Penning-Fallen-Aufbau SHIPTRAP der Gesellschaft für Schwerionenforschung (GSI) in Darmstadt
15.11.2007 BIS 31.12.2007	Forschungsstipendium für „rp-Prozessrechnungen für SHIPTRAP“
01.10.2006 BIS 01.04.2007	<b>Forschungsaufenthalt am National Superconducting Cyclotron Laboratory der Michigan State University in East Lansing, MI, USA.</b> Aufgabenbereiche: Computernetzwerkrechnung in der astrophysikalischen Kernphysik zur Bestimmung des Einflusses von Messungenauigkeiten von atomaren Massen auf die astrophysikalische Elementerzeugung, Aufbau eines Computer-Clusters sowie <b>Erstellung der Diplomarbeit</b> „Influence of Mass Uncertainties of Exotic Nuclei on the rp- and vp-Process Model“
JUNI 2006	Experiment zur atomaren Massenbestimmung am Penning-Fallen-Aufbau SHIPTRAP der Gesellschaft für Schwerionenforschung (GSI) in Darmstadt